AFRL-IF-RS-TR-2006-294
Final Technical Report
September 2006

SEMANTIC WEB DEVELOPMENT

Massachusetts Institute of Technology (MIT)/World Wide Web Consortium (W3C)

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. K535

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

STINFO FINAL REPORT

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Rome Research Site Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-IF-RS-TR-2006-294 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/s/

RANDALL J. MCINTYRE
Work Unit Manager

/s/

JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
Sep 06	Final	Aug 00 – Dec 05

4. TITLE AND SUBTITLE
SEMANTIC WEB DEVELOPMENT

5a. CONTRACT NUMBER
F30602-00-2-0593

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62301E

6. AUTHOR(S)
Tim Berners-Lee, Ralph Swick

5d. PROJECT NUMBER
DAML

5e. TASK NUMBER
00

5f. WORK UNIT NUMBER
15

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Massachusetts Institute of Technology (MIT)/World Wide Web Consortium (W3C)
77 Massachusetts Avenue
Cambridge Massachusetts 02139

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFSA
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2006-294

12. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. PA# 06-681

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The work funded by DARPA and done by MIT and W3C under DAML Agent Markup Language (DAML) project between 2002 and 2005 provided key steps in the research in the Semantic Web technology, and also played an essential role in delivering the technology to industry and government in the form of open W3C standards. The Web Ontology Language (OWL), a W3C Recommendation, is widely adopted in industry and academia and standardization work has started on a Rule Interchange Format (RIF). The chief products of research were the development of the SWELL logic language, instantiated as the Notation3 language; the experience of its use, and the software tools such as cwm developed around it. These tools, released as open source, have formed an on-ramp for many newcomers to the semantic web technology. Despite these successes, the Semantic Web has a long way to go - both in terms of research on the higher layers and in deployment. The report discusses factors which may have affected deployment speed, and concludes with an outline of ongoing efforts which would be appropriate.

15. SUBJECT TERMS
Semantic Web technology and development, OWL, SWELL logic language, Notation3 language, cwn

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	UL	66	Randall J. McIntyre
U	U	U			19b. TELEPHONE NUMBER (*Include area code*)

Table of Contents

1. 0 Introduction 1
    1.1 Semantic Web: Definition, motivation and goals 1
2.0 Results and Discussion 3
    2.1 SWeLL, or Notation3 3
      2.1.1 Motivation special to this environment 3
        2.1.1a Objective computation over the contents of information resources 4
        2.1.1b No Closed world 4
        2.1.1c Involve the Web 4
      2.1.2 Success criteria 4
        2.1.2a Objective computation over the contents of information resources 4
        2.1.2b No Closed world 4
        2.1.2c Involve the Web 5
        2.1.2d N3 Logic 5
    2.2 The cwm software suite 5
      2.2.1 Basic platform 5
      2.2.2 Parsing and Serialization 7
      2.2.3 Store 7
      2.2.4 Inference engine 7
      2.2.5 Lists 7
      2.2.6 Sets 8
      2.2.7 Builtin functions 8
      2.2.8 Web-aware queries 8
        2.2.8a Definitive documents 9
        2.2.8b Remote query processing 9
      2.2.9 Report generation 10
3.0 Relationship with stated goals 10
4.0 Language development 11
    4.1 OWL development 11
    4.2 Rule language development 11
5.0 Tools 12
    5.1 SemWalker & Ontaria 12
    5.2 Haystack 12
    5.3 Tools: Annotea & Algae 13
6.0 W3C Testbed 13
    6.1 W3C At A Glance 14
    6.2 Technical Reports Index 14
    6.3 Teleconferencing 14
7.0 Application Integration: Personal Information Management 15
    7.1 Calendar Data 15
    7.2 Handheld Device (PDA) Synchronization 16
    7.3 Travel Itineraries 16
    7.4 Addressbook and Contact Information 17
    7.5 Personal Finances 17

7.6 Music Collections 17
7.7 Issue tracking with Internet Mail 17
7.8 Photos, calendars, and maps 17
7.9 Problem: sensitivity of test data 18
7.10 Database transition 18
8.0 Summary 19
8.1 Semantic web growth: speed determining factors 19
8.1.1 The Complexity of the Technology 19
8.1.2 Query language 19
8.1.3 Incubator community 19
8.1.4 Generic browser 20
8.1.5 Dereferencable symbols: browsable data 20
8.2 Upcoming needs 20
8.3 Organizational note 20
9.0 References 21
10.0 Appendices 27
10.1 Appendix A 27
10.2 Appendix B 32
10.3 Appendix C 44
10.4 Appendix D 49
10.5 Appendix E 53
11.0 List of Symbols, Abbreviations and Acronyms 59

List of Figures

Figure 1: Roadmap diagram: the layer cake **1**
Figure 2: The Semantic Web Wave **2**
Figure 3: Cwm circles and arrows software modules **6**
Figure 4: Helsinki itinerary **16**
Figure 5: GPS data mapping **18**

List of Tables

Table 1: Documentation systems and Semantic Web techniques **15**

1. 0 Introduction

1.1 Semantic Web: Definition, motivation and goals

The DAML project was an important effort by DARPA driving an early step in the development of the Semantic Web. Specifically, in the much-quoted 'layer cake' roadmap,

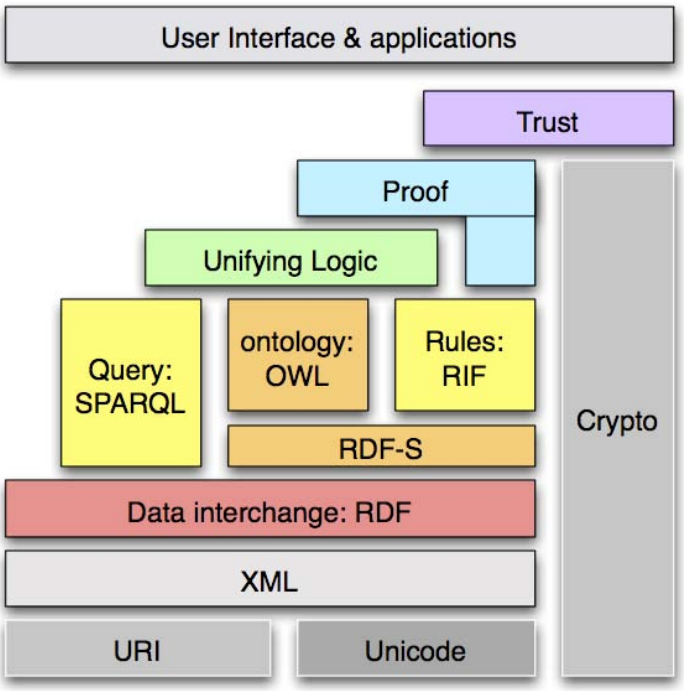

Figure 1: Roadmap diagram: the layer cake

the step is the development from research project to common widely shared standard of the Web Ontology Language, initially referred to as the DARPA Agent Markup Language. The roadmap calls for languages of various levels of expressivity to be developed in such a way that information expressed in these various languages can be shared, and can express things about the same unbounded set of things and concepts. This sharing is enabled by using the global URI address space to refer to all things and concepts.

During the project's lifespan, different layers of the cake required different levels of work. The XML, URI, Unicode and RDF layers were all already well established standards. The DAML/OWL layer had been the subject of many research projects in the US and Europe, and the task addressed was to come to a clear understanding of the types of logic involved, and bring it to the level of a common standard. Meanwhile, the higher layers such as rule and query languages were the subject of research but also wide deployment of non-standard languages which were not yet deemed ready for standardization. The layers such as proof exchange and policy-aware and transparent accountable systems ("trust") continued to exist only at the research level.

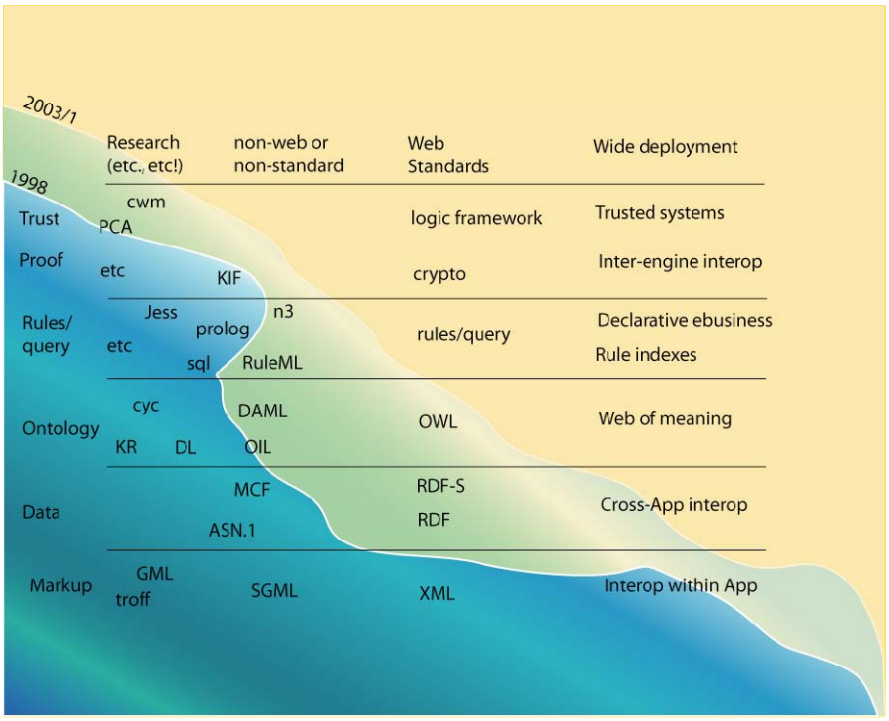

Figure 2: The Semantic Web Wave

The project described here involved participation in test activities as appropriate levels. A software platform was developed as a workbench for research, as a test implementation of standards, and as a openly available (open source) toolkit to promote the technology by making an easy on-ramp for developers new to the field. Also, various test scenarios were implemented to test that the concepts of integration of data between diverse applications were in fact practical. These tests were typically based either on the personal information management space, or on the enterprise automation space, using the World Wide Web Consortium (W3C) itself as the test enterprise.

The eventual success criteria for the project, indeed for the DAML work as a whole, will be in the long term a strong wide and generally adopted semantic web technology which provides humanity with the ability to reuse data and logical information in an unprecedented way. Only in perhaps a decade we will be able to judge this transition retrospectively, but we can at this stage (2006) note that the roadmap has indeed been followed, with new layers moving from research to standardization every few years. Some of the factors affecting this speed are discussed below.

At the immediate scale, the success criteria can be judged in terms of the development of the software as outlined in the proposal, and the creation of languages, such as the Semantic Web Logic Language (SWeLL), also described in the proposal. This software and the languages developed are inextricably interconnected in terms of mutual influence. In this report we take the arbitrary choice of discussing the languages first, then the software platform.

2.0 Results and Discussion

2.1 SWeLL, or Notation3

The proposal [Proposal] called for the development of a suitable logic language for use on an open and unbounded semantic web. This language, described as SWeLL in the proposal, was actually delivered as a language now known as Notation 3.

This is a language which is a compact and readable alternative to RDF's XML syntax, but also is extended to allow greater expressiveness. It has subsets, one of which is RDF 1.0 equivalent, and another which is RDF plus a form of RDF rules.

N3 is described in more detail elsewhere, and so here we give only an overview of its role in the project and in the goals of semantic web development. The N3 home page [N3 home] is a general introduction and a center for other resources. The developer learning N3 is invited to try the tutorial, [N3 tutorial] while implementers looking for a particular detail of the definition of the logic are steered toward the operational semantics [N3 logic] There is also a list of other N3 resources [N3 Resources].

The aims of the language were to:

- optimize expression of data and logic in the same language;
- allow RDF to be expressed;
- allow rules to be integrated smoothly with RDF;
- allow quoting so that statements about statements can be made, and to be as readable, natural, and symmetrical as possible.

The language achieves these with the following features:

- URI abbreviation using prefixes which are bound to a namespace (using @prefix) a bit like in XML;
- Repetition of another object for the same subject and predicate using a comma ","
- Repetition of another predicate for the same subject using a semicolon ";"
- Bnode syntax with specific properties just put the properties between [and]
- Formulae allowing N3 graphs to be quoted within N3 graphs using { and }
- Variables and quantification to allow rules, etc. to be expressed
- A simple and consistent grammar.

2.1.1 Motivation special to this environment

The following needs were seen initially as driving the development of the system, as a result of its being intended for communication between agents whose only shared context is an unbounded web of information. We designed the system such that all data sources were considered as having a (possibly empty) N3 semantics: that is, for each resource on the web there is an N3 formula which expresses at least a subset of the intent of the publisher of that resource.

2.1.1a Objective computation over the contents of information resources

On the web, it is of course quite unreasonable to believe everything one reads. So it is on the Semantic Web: an agent may read many things, but in general it is aware always of the provenance of data. We started the project convinced that the many systems which roughly divide agents or data into "trusted" and "untrusted," or gradations thereof, were fundamentally naive. In practice, we demanded the ability to trust a certain source for statements of a certain form only under certain conditions, or even to deduce a separate fact from the presence of statements on in a specific source (e.g. "If MIT says x is a student then x is a human being").

2.1.1b No Closed world

The systems we build are designed to operate in an open unbounded web of data and logic. At no time does any agent in such an environment ever consider that it knows everything that has been said about anything. Therefore the Closed World Assumption (CWA) in which a system is allowed to assume something is false if it not manifestly true from the given data, becomes quite unusable as it stands. Any such assumption, if it is to be shared with other agents, can only be shared if the actual scope of the data involved in the assumption is well defined.

2.1.1c Involve the Web

The Web is seen, from the point of view of an agent processing N3, as a function which maps a URI into its N3 semantics, that is, an RDF graph or rather in general, an N3 formula. It is clearly necessary to make this function available for use in rules and information processing in general.

2.1.2 Success criteria

The success of N3 as a result of the DAML project can be judged as follows:

1. Within the project, in the testbed applications, N3 was found to be sufficiently powerful to meet its requirements without further extension.
2. The simplicity of the syntax has made it the language of choice in examples in documentation, email discussions and Internet Relay Chat (IRC) discussions across the Semantic Web development community.
3. Parsers and processors have been written in many programming languages.
4. Subsets of N3 have been adopted in NTriples [NTriples] a simple test format for RDF data, and Turtle [Turtle], a subset of N3 which excludes nested formulae, but can express any RDF graph.

The N3 style syntax has been used within other languages designs, such as SPARQL [SPARQL], the standard RDF query language under development.

We now consider the requirements of operation in a semantic web context are met.

2.1.2a Objective computation over the contents of information resources

This requirement forced the extension of RDF to N3 so as to include quoted graphs within a graph. These are known as N3 formulae. The `log:includes` function expresses the constraint that one formula includes (strictly, N3-entails) another.

2.1.2b No Closed world

To meet this requirement, there is no CWA form of negation. The only negations available are the negative forms of the built-in operators. Importantly, this includes the `log:includes` operator which has a negative form `log:notIncludes`. This allows one to make a condition that something has not been said by a given source. For example, "If the order form gives no color,

the color is black". This is the open non-CWA form of the unacceptable: "If there is no color then the color is black".

2.1.2c Involve the Web

The web function is given the URI `log:semantics`. The software platform evaluates it in real time.

Also, the cwm system can operate in modes such that when a symbol occurs during processing, it will be dereferenced on the web so that any published information about the symbol may be loaded. This is described in more depth below.

2.1.2d N3 Logic

The N3 design allows logical operators to be introduced simply as new properties, just as RDF allows new languages (effectively) to be designed just by the introduction of new RDF Properties. While the logical operators may then be simply regarded as an ontology for logic, in fact the language equipped with these operations becomes, effectively, a logic language, which we refer to as N3 logic.

2.2 The cwm software suite

Cwm (pronounced ‘koom’) is a general-purpose data processor for the Semantic Web, somewhat like sed, awk, etc. for text files, or XSLT for XML. It is a forward chaining reasoner which can be used for querying, checking, transforming and filtering information. Its core language is RDF, extended to include rules, and it uses RDF/XML or RDF/N3 (see Notation3 Primer [N3 Primer]) serializations as required. Cwm is written in python [Python]. Like all the code developed in this project, it is released under an open source license, the W3C Software License.

2.2.1 Basic platform

The cwm platform is a general purpose Semantic Web development platform written in Python. It has been used as a library with a python API, but its chief goal is to serve as a general purpose data manipulation tool for Semantic Web languages, just as sed, awk and grep served as basic data manipulation tools in a unix command line-oriented environment. It is also designed as a proof of concept, and a platform for proofs of new and varied concepts. To this end, the emphasis was on extensibility, and not on optimization for speed. The result has been that some projects have been started which conveniently used cwm, but which took longer and longer to run as the data sizes grew. This has lead to pressure to produce cwm-compatible systems which run faster, and we note that work is in progress to integrate the rete-based pychinko [Pychinko] reasoner with the cwm code itself.

The software module high-level structure is shown below.

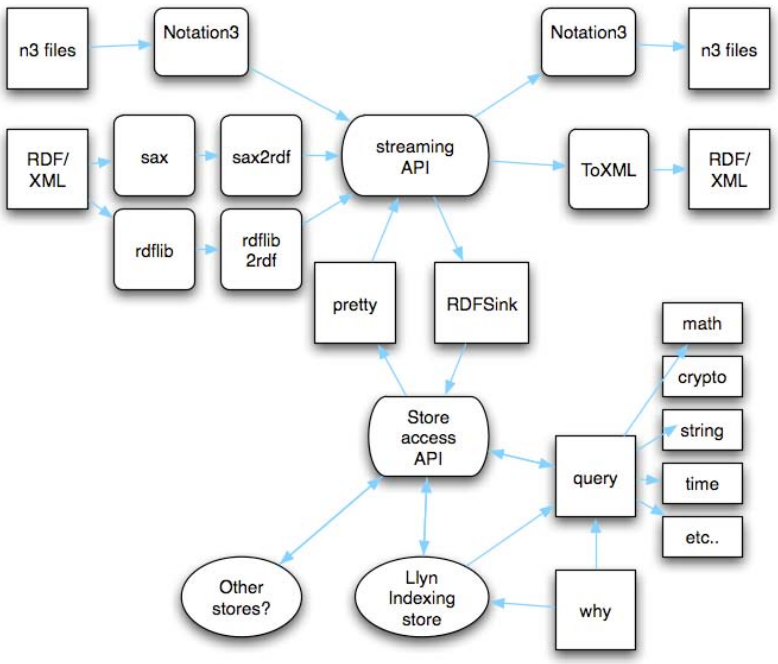

Figure 3: Cwm circles and arrows software modules

2.2.2 Parsing and Serialization

The N3 and RDF languages are grammatically similar, to the extent that it is possible to translate one into the other in a streaming mode. The efficiencies evident from this mode of operation led to a design of an abstract syntax interface which parsers and serializers supported. This is still in use, but experience was that as the N3 language became more sophisticated, the streaming interface became more burdensome to support as an output format for the serialization module.

2.2.3 Store

The store (llyn.py) is a triple store, extended to cover the full N3 language, and also record the provenance of each statement. A statement therefore is stored as the subject, predicate and object of RDF, plus the '*context*,' defined as the identity of the formula in which the statement is found, and a record of the reason object which stores the reason for the statement having been added to the store.

The store was originally designed and built with four indices, so that a list was kept of each statement in which a given term occurred as, respectively, the subject, predicate, object or context. When there was a need to improve performance, this was changed so that now 7 indexes are used, as every combination of subject, predicate, and object pattern with wildcards are indexed. This of course increases the time to load the store, and it is questionable whether the improvement in access is worth the effort of indexing. A design goal was to make the store module switchable so that tasks with specific profiles could use the most efficient form of store, but this was not given priority.

2.2.4 Inference engine

The inference engine is at heart a simple forward chaining reasoner for rules. A rule is stored as a statement whose predicate is `log:implies`. in the form

```
{ ?x parent ?y. ?y sister ?z } log:implies { ?x aunt ?z }.
```

The matching engine which runs the rules operates simply by recursive search, with some optimizations. Firstly, the rule set is analyzed to determine which rules can possible affect the output of which other rules. A partial ordering is found, which for example in some cases will produce a pipeline. Otherwise, where rules can interact, they are tried repeatedly until no further rule firings occur. The second optimization is that, when matching a graph template, which is a series of template statements, the statements are ordered for processing as a function of the length of the index which would have to be searched, doing the smallest indexes first. This provides a significant improvement in many real-world examples where the data is very asymmetrical, in that some areas of the graph are dense and tabular in form, and others sparsely connected.

The inference engine also performs two extensions to its normal role, these being the execution of built-in functions, and the delegation of parts of the query to remote systems or remote documents.

2.2.5 Lists

Cwm uses rdf collections extensively, and puts a stronger interpretation than RDF, so that they can be used as triples. The cwm system assumes:

- Any two things which have the same rdf:first and rdf:rest are the same (owl:sameAs).
- If one thing has rdf:first x and rdf:first y then x and y are the same.
- All lists exist, so the statement that there exists a list with a given set of elements is itself tautological and so can be ignored

Lists work, therefore, as a compound data type. (Formula is another compound data type). They can contain variables. They are used, among other things, as argument lists for Nary functions, for example:

```
( 1 2 ) math:sum 3.
```

Cwm does not use the rdf concepts of bag, and sequence. These were found to be awkward in that their reification as RDF uses an infinite set of properties such as rdf:_1, rdf: 2, etc.; and deficient in that there is no record of number of members of the bag.

2.2.6 Sets

There is some support for sets. We believe that distinguishing between ordered lists and unordered sets is in general going to be very important for data on the semantic web. Too often, XML documents leave the question of whether the order of data is important as implicit and not always understood. Too often, in N3, we are tempted to use a list because of its simple LISP-like syntax, when in fact we are describing an unordered set. However, operations on unordered and ordered sets are quite different. We use `owl:oneOf` to create a set as the class which has just the members of a list as members. (This form of set is used for the reification of N3, when describing the sets of variables and statements associated with a formula.)

2.2.7 Built-in functions

Many reasoning engines have an ability to perform arithmetic, but arithmetic facts are separated out from the facts of the knowledge base as being fundamentally different things. In cwm, this is not the case: arithmetic facts, as with all other facts where the validity can be checked or the result evaluated by machine, are represented as RDF properties. This is done for various reasons. Philosophically, the design was not prepared to commit to a partitioning of knowledge into two parts in this fashion. It was felt necessary to be able to reason about these functions as well as to evaluate them. We understand that this causes problems for a Description Logic based systems, but this is not a DL system. It was also done for architectural simplicity. Making this simple decision allows all the language support for RDF and N3 to be immediately adopted for the arithmetic expressions; the store can store them, the serializer can output them and so on. Contrast this with the situation in for example the SPARQL [SPARQL] language, in which a special place in the query expression is reserved for filter expressions, and a whole separate syntax is supported for it.

Within the reasoner, built-in functions are made part of the query. During query optimization, *light* built-ins (such as negation of an integer) are assumed to be faster than searching the store, and are performed the moment they can be, while *heavy* built-ins, such as accessing the Web, making a remote query, or recursively invoking the reasoner itself, are assumed to take a long time, and are postponed until anything faster, including searching the local store, has been done.

Nary functions are implemented using lists, as mentioned above. This was found to be very satisfactory.

2.2.8 Web-aware queries

The cwm system can operate in modes such that when a symbol occurs during processing, it will be dereferenced on the web so that any published information about the symbol may be loaded.

- When x occurs as a predicate in a loaded statement
- When x occurs as a subject in a loaded statement
- When x occurs as an object in a loaded statement
- When x occurs as a predicate in a query template, or is bound to a variable in that position, while a query is being matched against the knowledge base.
- When x occurs as a subject in a query template, or is bound to a variable in that position, while a query is being matched against the knowledge base.
- When x occurs as a object in a query template, or is bound to a variable in that position, while a query is being matched against the knowledge base.

These may in the software be enabled individually. They operate recursively. The first form, looking up predicates, creates what we call the Ontological Closure of a graph. It is the case in practice that the ontological closure of documents is finite, and contains useful information.

The second and third forms are, in general, not guaranteed to terminate except within a deliberately designed dataset with no links to the public Semantic Web in general. They were not found to be very useful.

The fourth, fifth and sixth modes quickly look up new variables as they occur in query processing and do not suffer from the tendency to bring in the entire web. A query is evaluated by matching a graph template to the graph of data which can be found on the web. If each node in the graph has useful and relevant information associated with it, and that information is loaded by the query engine when the node is first mentioned, then it is possible for the query to successively bring in documents which together form the parts of a large graph as it is needed.

These modes of operation, and ones like them, are, we believe, important to the development of the Semantic Web. Future research should be directed toward protocols which involve conventions for the sort of information which is published against the URIs used as symbols for arbitrary things in the Semantic Web.

2.2.8a Definitive documents

There are times when a particular document on the web contains all cases of a particular relation. For example, the relationship between a US state and its two letter code exists in 50 cases. Another document might, for example, store a definitive list of the MIT course numbers.

In this case, queries involving these properties become self-answering according to the following protocol. The query processor looks up the ontology for the property when it finds it in a query. The ontology file mentions that there is a definitive document for the property, with a statement like, for example:

```
state:code log:definitiveDocument <stateCodes.rdf>.
```

The client then converts any query or query part of the form `?s state:code ?y` into a query on that document.

2.2.8b Remote query processing

The modes mentioned above allow cwm to pick up data while it is processing a query, by loading RDF or N3 documents. We were also interested in applications in which large quantities of existing data were in live SQL databases. The cwm query engine has the ability to pick up metadata from the schemas (the ontological closure above) which directs it, for certain specific

properties, to convert that part of the query into an SQL query. This is done by making the assertion that the property has a `log:definitiveService` whose URI is a form of mySQL [mySQL] URI which carries the information on how to access the data.

The implementation is a proof of concept only; it operates only with mySQL databases. We would recommend now (2006) that SPARQL will soon be available as a standard, and future designs implement this functionality by converting the query into SPARQL. In this way, systems of federated SPARQL servers should be set up. This is a very interesting direction for future research, specifically the protocols for defining the conditions under which a given server should be contacted for a given form of query.

2.2.9 Report generation

A practical need in semantic web systems is for human readable output, most typically for web-based use as either xHTML for text documents, or SVG for diagrams. A simple hook was added to cwm to allow report generation using rules, as follows:

The rules generate statements of the form

```
?k log:outputString ?s.
```

where `?k` is a key giving the ordering of the output, and `?s` is a string, typically a fragment of XML source. The rules are then run in a cwm session which uses the --strings argument to select string output rather than N3 or RDF serialization.

Another sometimes very effective way of presenting relatively small amounts of interconnected data is as a graph. The ATT GraphViz [GraphViz] open source program automatically draws graphs. We developed a small ontology equivalent to the GraphViz input format, so that N3 rules could be used to generate graphs, including a style ontology to define the mapping between classes of object and the color and shape of displayed nodes. At least one hardened XML opinion leader was converted to the RDF world just by the ability to display such graphs.

3.0 Relationship with stated goals

The original proposal mentions the development of tools:

- Semantic content authoring tools
- SWeLL-based proof generators
- SWeLL-based proof validators
- Access control: controlling access to documents on various parts of the W3C site
- Personal INformation Schema (PINS) controls: using DAML and SWeLL, users will be able to attach conditions to use and re-use of information they contribute to the Semantic Web

The first goal was effectively met by the fact that N3 was designed as a very easily writable language. In almost all cases the data needed, as well as the rules, was simply authored by hand in N3. In addition to this, we put significant effort into the conversion of non-RDF data into RDF. This was important from the point of view of feasibility of deployment of semantic web systems on top of existing data, so experience over a wide range of formats is important. It also encouraged various communities to start using RDF technology. This work is elaborated on below.

The proof generation facility is an integral part of the cwm software. For a cwm command line which would have produced a given result, adding --why to the command line produces instead a proof of that result.

A separate program, check.py, is used to validate a proof. The intention was for check.py to share as little code with the proof generator as possible, but the built-in functions and the unification algorithm were not worth coding independently.

The access control scenario was modeled, although it was not installed on the live W3C web site. The access control scenario implemented was one using delegated authority implemented using public key cryptography. This drove the development of cryptographic builtin functions.

The use of RDF to define definitions of policies around the use of code is something enabled by the cwm functionality. During the period of the DAML project we did not develop these areas as far as we would have liked. However, we are, under future funding (Policy Aware Web (PAW [PAW]) and Transparent Accountable Datamining Initiative (TAMI [TAMI])), implementing these ideas much more fully.

4.0 Language development

4.1 OWL development

DAML and OIL languages were developed during this time, in the United States and Europe respectively, and the results transferred to the W3C to form the Web Ontology Language (OWL). This was the major thrust of the program, and the transfer and successful development of the language at W3C into a global common interoperable standard for government, academia and industry may be considered be the principle claim to success of the DAML program.

We actively participated in all stages, in the design of the language (both in the ad hoc joint committee and the later W3C working group) and the coordination and oversight of the process of building consensus around the new language at W3C. The commendable vision and leadership of the DARPA program managers, particularly the initial P.M., Prof. J. Hendler, provided that the effort would be sufficiently resourced through these stages without loss of momentum.

OWL became a W3C Recommendation on 10 February 2004.

4.2 Rule language development

The N3 language developed by this project demonstrated the power of rule-based systems in processing semantic web data. It also demonstrated that, when expressed in a suitably architected semantic web language, rules themselves could have the same ability to be shared as data has when expressed in RDF.

The status of rule languages at the time was that a group of those interested had already worked for several years on RuleML, a common but non-standard interchange language. The project provided effort in analysis of the state of development and the relationships of various products, and, through meetings and a W3C workshop [Rules WS], the bringing together of academics from the DAML community among others, and industrial rule-based system providers. This has resulted in the chartering of a working group aiming to produce a standard language which will meet the requirements of all the players. This is a challenging task, perhaps even more

challenging than the OWL development, but we are optimistic. We note, however, the absence of ongoing DARPA support comparable to that of the OWL work.

5.0 Tools

5.1 SemWalker & Ontaria

The Semantic Web we design is very declarative in nature, and a natural choice of implementation language would appear to be prolog. The SemWalker tool is a implementation in prolog of the basic RDF library functionality, including parsers, store, and query. SemWalker was created in order to provide a general user interface platform for directly browsing Semantic Web data.

The SemWalker platform was used to create a web-based portal for viewing ontologies, known as Ontaria. The system included prolog-based Semantic Web crawler and indexer.

SemWalker [SWalker] is a toolkit for building browsers for Semantic Web data. SemWalker includes a data store and an RDF/OWL harvester. It is intended to run as a server-side application and implements user interfaces via HTML so that users use common Web browsers to connect to SemWalker applications. SemWalker has some preliminary work to support specialized views of objects based on their (RDF) Class.

Ontaria [ONT] is a searchable and browsable directory of OWL ontologies built on SemWalker. The focus of Ontaria is on people who are creating RDF content and wish to find existing vocabularies they can use. The views implemented in Ontaria are customized for the purpose of browsing OWL ontologies.

The results of the SemWalker work provided insight into human interface issues for generic Semantic Web browsers. The architecture SemWalker used (generic browsing with specific more crafted views available for special cases) is the basis of later work on browsers such as PiggyBank [Piggy Bank] and Isaviz [Isaviz] which are driven by the embryonic *Fresnel* [Fresnel] language, and by our own later Tabulator [Tabulator] browser.

Development of the Semwalker prolog-based platform was discontinued, partly due to lack of a community of prolog users who would benefit from the open source code, but mainly in order to prioritize work on a standard rule language.

5.2 Haystack

The Haystack project [Haystack] has been focusing on the development of general purpose user interfaces for arbitrary Semantic Web data. Any individual may contribute new types of information to the Semantic Web, and it will be difficult, if not impossible, for application developers to produce visualizations for all those new information types, especially given the many different uses to which individuals may wish to put the information they have gathered from multiple semantic web sources. Therefore, we have begun developing frameworks to let end users of semantic web information specify appropriate visualizations, and more generally to create their own information management applications over the Semantic Web, choosing precisely which information objects they want to work with and how they want to view and manipulate those objects. Such “end user application development” would let end users create

workspaces designed specifically to solve their specific information management problems. Our approach combines three elements:

- A workspace manager that lets users specify the information objects that they want to lay out to work with in their application and the operations that should be applicable to those information objects;
- A view manager that lets users specify how each of the information objects in their workspace should be shown---what properties of those objects they want to see, and how they should be laid out; and
- A channel manager that lets users specify queries that dynamically maintain collections of information relevant to the task

Rather than specifying views, workspaces, and channels programmatically, users put them together using natural visual operations such as dragging, dropping, and resizing, that they are already familiar with as tools for managing their desktop environments. The workspaces, views, and channels designed by these end users are themselves represented using RDF in the Semantic Web, creating an opportunity for users to share the views and workspaces that they design with others, and for unsophisticated users to craft their "applications" by tweaking preexisting ones instead of creating them from scratch.

We have implemented our system as part of the Haystack information management platform. This tool provides a set of cooperating technologies and tools supporting end-user creation, visualization and manipulation of Semantic Web content, as well as application development for the Semantic Web.

5.3 Tools: Annotea & Algae

Annotea [Annotea] was an experiment in shared, collaborative annotations of Web documents using RDF. It provides a basic metadata query protocol for asking multiple servers for metadata about a named resource. In the case of Annotea, the named resource is a Web document and expected metadata is RDF describing annotations concerning that document. The user interface to Annotea was integrated into the W3C Amaya [Amaya] editor/browser. The form of annotation RDF data supported by the Amaya interface is HTML text with pointers to specific segments of the annotated document. Annotea annotations are stored externally to the annotated documents; write access to the documents is neither necessary nor expected.

The Annotea metadata query protocol is tuned to allow clients to issue a simple HTTP request to the metadata server to request the known metadata about a given Web document. In conjunction with Annotea we designed and implemented the Algae [Algae] query language and persistent RDF database to be a general-purpose query language for RDF. Underlying the simple access protocol of Annotea, the Algae system provides full-function data store and triple pattern-matching query interface. The Algae work was important input to the W3C Data Access Working Group and was incorporated into the W3C standard SPARQL [SPARQL] Semantic Web query language for RDF.

6.0 W3C Testbed

The World Wide Web Consortium (W3C) is an industry consortium with 400 member organizations (vendors, research organizations, and end-user organizations) participating in 38

working groups. These working groups publish their formal output as W3C Technical Reports. Regular news bulletins and periodic reports to the Membership inform the Members of the status of the work in each group. The formal workflow of the Consortium lends itself nicely to be a testbed for Semantic Web deployment.

6.1 W3C At A Glance

W3C At A Glance [W3C Glance] was an early aggregator of information in RSS/RDF form from various parts of the W3C, presenting a view of related information using a hierarchical structure gleaned from the data. A novel feature of *W3C At A Glance* was its integration with the RDF-based page access control system deployed on the W3C web site. Users of *W3C At A Glance* are presented different aggregate views of the information based on their access rights to the original sources of the data. *W3C At A Glance* was separately funded, but the work was conducted in close collaboration with the DAML Semantic Web Development project.

6.2 Technical Reports Index

The W3C Technical Reports index [W3C TR] lists the formal work product of each W3C Working Group categorized by one of six "maturity levels". The maturity levels indicate the state of the work, from (first) working draft to adopted standard (called "W3C Recommendation"). The W3C Process [W3C Process] defines the formal steps necessary for a document to advance in maturity level.

For many years, the Technical Reports index had been maintained manually and the critical workflow data that it contained was not available in machine processable form. Checking the prerequisites for each maturity advancement was entirely manual. As part of a separately-funded workflow automation project, the W3C pages listing the Working Groups, their chartered time periods, their deliverables, and the Technical Reports index itself were all turned into authoritative sources of Semantic Web data by adding semantic markup to the existing HTML and using the following ontologies:

- org.n3 [org.n3]-- organizational structure
- roadmap [Roadmap] -- project dependencies
- doc.n3 [doc.n3] -- document management - versioning etc.

Data extraction tools were then written to allow cwm to check some of the dependencies and prerequisites at the time a new document is proposed for publication. The W3C Technical Reports index is now built automatically by cwm and is available to users in multiple views, as well as in machine-processable RDF form, allowing a variety of analyses to be performed.

6.3 Teleconferencing

W3C conducts all of its Working Group business by teleconference and simultaneous text chat (irc [IRC]). The collaboration metadata around teleconferences is data that can be merged with other W3C enterprise data; the teleconference calendar is published as dynamic RDF data, as is data about in-progress teleconferences (active participants and agenda state). A Semantic Web-enabled agent, Zakim [Zakim] was implemented which participates in the teleconference through the shared text chat. Zakim assists the meeting chairperson by reporting arriving and departing

participants in real time, accepting requests for agenda items, and tracking 'hand raising' requests to address the meeting.

7.0 Application Integration: Personal Information Management

The Web integrated a variety of documentation systems so that now we can follow links from tutorial articles to reference documentation, across organizations, with one click. But for data, we are still pre-Web. Airline reservation systems are online, but when you book an itinerary, getting the data to your personal calendar application often involves manually copying each field across. We explored the use of Semantic Web techniques to address these problems.

Topic	Format	Ontology	RDF Tools
Calendar	iCalendar (.ics)	RDF Calendar	toIcal.py [toIcal], fromIcal.py [fromIcal]
Handheld PDA	Danger OS XMLRPC	palmagent/danger	palmagent [PalmAgent]
	PalmOS .db files	palm calendar, datebook	
Handheld GPS	Garmin		fromGarmin.py [fromGarmin]
Travel Itineraries	telex	various	grokTravItin.pl [TravItin], cityLookup.n3 [cityLookup],
Address book/Contact Information	vCard	swap/contact [swap/contact]	mso2vcard.n3 [vcard]
Personal Finance	OFX, QIF	swap/fin	qfx2n3.sed [qfx2] qif2n3.py [qif2]
Photo metadata	EXIF		jhead [jhead] (as adapted)
Email	RFC822	swap/email	mid_proxy.py [mid.proxy], aboutMsg.py [aboutMsg]
Software dependencies	makefile		make2n3.py [make]
	dpkg		fink2n3.py [fink]

Table 1: Documentation systems and Semantic Web techniques

7.1 Calendar Data

The IETF proposed standard for calendar information is iCalendar (RFC 2445 [RFC 2445]). It is supported by Apple's iCal and similar applications.

We developed:

- an ontology corresponding to the IETF iCalendar standard, derived by machine from the text of the standard
- conversion tools from iCalendar syntax to RDF/XML (ical2rdf.pl [ical2] and fromIcal.py [fromIcal])
- conversion tools from RDF/XML to iCalendar syntax (toIcal.py [toIcal])
- a collection of tests and an automated harness to check the conversion tools
- a tool to convert the hCalendar [hCal] microformat to RDF/XML (glean-hcal.xsl [glean])

A number of collaborators joined in development of these tools. They are about as robust as other state-of-the-art iCalendar tools; that is: some issues around time zones and recurring events remain, but for most uses, they work well.

See:

- Dan Connolly and Libby Miller *RDF Calendar - an application of the Resource Description Framework to iCalendar Data* [RDF Cal] W3C Interest Group Note 29 September 2005

7.2 Handheld Device (PDA) Synchronization

We explored import/export and synchronization of data from Palm OS devices and Danger Hiptop devices in the palmagent [PalmAgent] tools.

We developed fromGarmin.py, which combines with pygarmin to produce RDF data from a GPS device.

7.3 Travel Itineraries

We developed grokTravItin.pl [TravItin] which converts data from SABRE telex format to RDF, as well as N3 rules to look up latitude, longitude, and time zones of airports, and used these in combination with calendar tools and PDA import tools. We used cwm's reporting features to write xearth files that can be used to visualize trips.

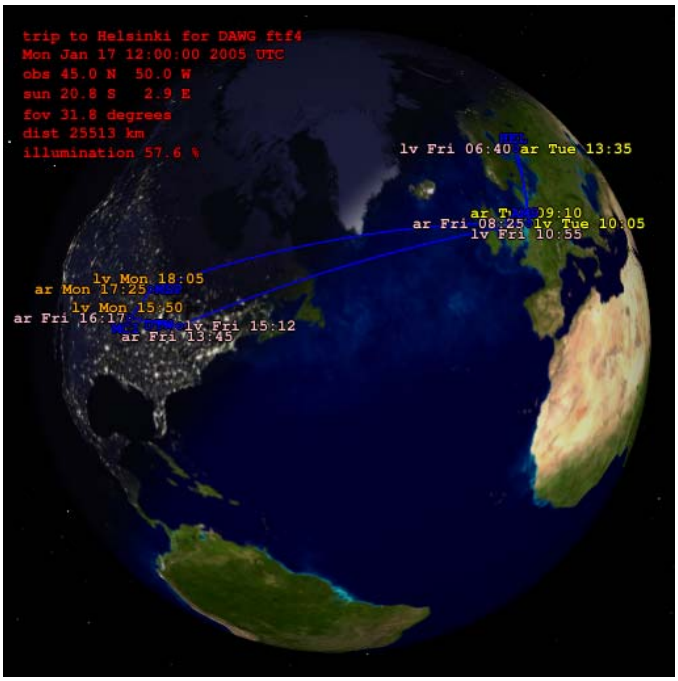

Figure 4: Helsinki itinerary

We presented this as part of our *Semantic Web Tutorial Using N3* [N3 Tutorial] at WWW2003 and WWW2004.

7.4 Addressbook and Contact Information

We developed lookout.py [lookout] a tool demonstrating how to export Microsoft Outlook data to RDF/N3.

7.5 Personal Finances

OFX is a format starting to be deployed by banking institutions to provide customer's data upon request. qfx2n3.sed [qfx2] is a demonstration of converting OFX (Open Financial Exchange [OFX]) data to RDF. qif2n3.py [qif2] is a demonstration of converting QIF (Quicken Interchange Format [QIF]) data to RDF. These were combined with a transcription of the IRS 1040 rules into N3 for preparing taxes.

We also experimented with converting Quicken reports to RDF calendar and .ics formats via the hCalendar microformat.

7.6 Music Collections

Apple iTunes music data are stored in Apple OS/X Plist (property list [Plist]) files (as are Safari bookmarks). We developed plist2rdf.xsl [plist2], which converts them to RDF.

7.7 Issue tracking with Internet Mail

We developed aboutMsg.py [aboutMsg] to convert email header data to RDF. We combined this with N3 rules for tracking cwm bugs as well as for tracking issues against the OWL and SPARQL specifications.

We also experimented with mid_proxy.py [mid.proxy] which demonstrates an IMAP to HTTP gateway, with RDF support.

7.8 Photos, calendars, and maps

In Making a map of the photos [Map Photos] we adapted the jhead tool to produce RDF and combined the resulting data with GPS data to plot photos on a map.

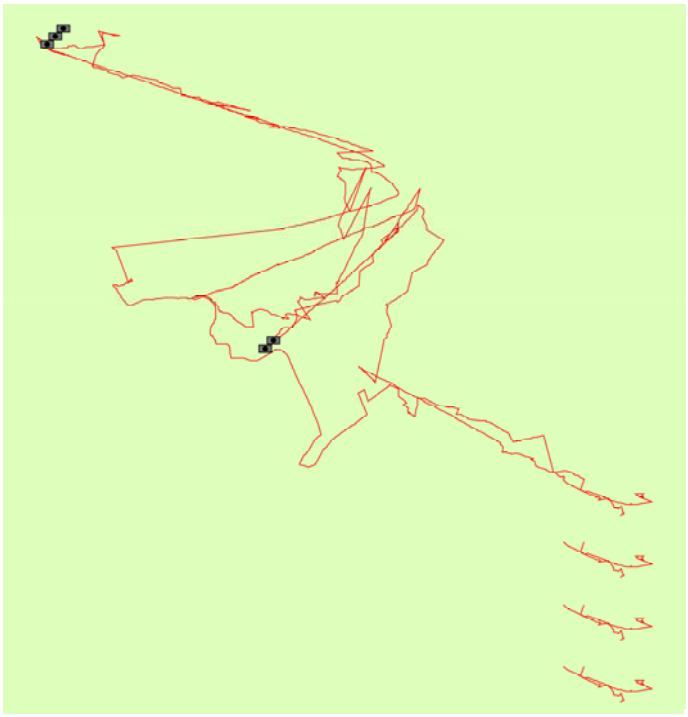

Figure 5: GPS data mapping

7.9 Problem: sensitivity of test data

Many of these tools were developed to address personal needs of the developers, who used the tools on the data from their personal lives. In only a few cases did we manage to publish anonymized data for others to use for demonstration and test purposes.

7.10 Database transition

Toward the goal of promoting the wide availability of semantic web data, we developed a stand-alone program to export an existing SQL database as RDF data on the web. Known as dbview.py, [DBView] this facility exports a series of interlinked RDF files, automatically generating URLs for all the objects (basically, rows) and properties (basically, columns) involved.

This facility was not given a lot of priority, being developed to the point that it demonstrated the feasibility of this technique. The hope was that database software vendors would adopt the idea, and provide their own products. There have indeed been several research projects such as D2RQ [D2RQ], and suggestions more recently that include commercial implementations.

We hope to develop dbview further in future projects, as we conclude that there is still a strong need for simple paths from SQL data to the Semantic Web.

8.0 Summary

We look at the effects of the work in two parts, the specific deliverables of the project and the overall goal of the promotion and furtherance of the Semantic Web as a powerful deployed technology.

8.1 Semantic Web growth: speed determining factors

Semantic Web technology is undergoing a steady uptake. A frequent question is why, in 1995, in its fifth year, Web technology was well deployed and generally understood by the public and yet in 2006, after 5 years, Semantic Web technology is not. There are several reasons that one might imagine would account for this.

8.1.1 The Complexity of the Technology

The HTML language as initially proposed was extremely simple. There were no particular mathematical properties it had to have, compared to the languages of RDF, OWL, SPARQL and RIF, in which choices had to be made between logics, and the languages needed to be carefully checked for consistency and appropriate power. This has been much more complex.

8.1.2 Query language

The technology layer above OWL consists of query and rule languages. The roadmap we put forward involved waiting until the data language (RDF) and the ontology language for defining terms (OWL) were settled before work started on query and rules. This is partly because of dependencies, but also partly because of limited resources. However, we have for the last 5 years been doing the equivalent of developing relational databases without SQL. The query language, SPARQL, now in the final stages of standardization, is, for many people, a key to the usability of Semantic Web data. The SPARQL work has not benefited from any US government funding.

8.1.3 Incubator community

Network systems, such as the telephone, electronic mail, the Web and the Semantic Web are subject to Metcalfe's law that the value of one component is strongly dependent on the number of other components. Such a system cannot get started until a critical mass of adopters has invested to populate a certain proportion of the system. DARPA's investment in the Semantic Web is an excellent example of this. The effort involved in getting, say, 10% of people to adopt a technology is much easier in a small community than a large one. The Web spread among High Energy Physicists, who were perfect early adopters. They had a major problem of disconnected information systems; they are bright, flexible people, and already had the most advanced technology in terms of workstations and networking. For the last 5 years, the Semantic Web has not benefited from this small incubator community. Now, however, it looks as though today's exciting leading edge scientific discipline, Life Science, is taking on that role, as judged from the deployment of ontologies and data, and the excitement at conferences and in interest groups. (It is possible that the defense industry may also be an early adopter, as it has strong needs, the ability and the resources, although the results of such work are often too sensitive to be available to inspire those outside.)

8.1.4 Generic browser

For the hypertext WWW, a growth enabler was the immediate benefit to anyone putting up HTML pages so that they could see the web pages in any browser. For the last 5 years, the Semantic Web has not really had a generic data browser. Certainly, there have been many browsers, but the powerful ones with an intuitive user interface have been specific to, and tailored for, applications longwell [Longwell] and CS AKTive [CS AKTive], while the generic ones have typically had limited property-list or circle-and-arrow graph views not capable of seriously providing a powerful view of large amounts of data. MIT hoped to undergo future work based on Haystack [Haystack] and the new Tabulator [Tabulator] project to make data immediately available in a compelling fashion, increasing the incentive to publish data, and also to allow early verification and correction of obvious errors.

8.1.5 Dereferencable symbols: browsable data

The development of the semantic web languages, particularly OWL, has been done under the architectural constraint that URIs are used for all global symbols. While this has indeed produced a language with the hoped-for qualities of supporting links between data and between ontologies, the custom in the community has been to develop project-wide data stores which are amassed from various sources, and then subjected to inference, query and visualization in-situ. The result has been that the individual URIs are not supported in the sense that one can take one, look it up and get a reasonable set of data about the object.

As on the HTML web, it is important to get reasonable information on dereferencing a URI with Semantic Web data. The actual value added by WWW or Semantic Web technology is the unexpected re-use of data. That can only occur if things are identified by URIs and these URIs are supported by servers serving appropriate data in RDF (and/or SPARQL query services).

8.2 Upcoming needs

In general, this analysis suggests that SW uptake will take place at a steady pace, but will be accelerated by funding in the areas of:

- powerful generalized UIs
- Seed data, such as basic science (properties of matter, chemicals, eTOC)

At this time it also seems appropriate to investigate the properties of the Semantic Webs we are planning to create on a large scale -- both webs of interconnected data and webs of rules from different rule sets.

8.3 Organizational note

The work described above was proposed by the W3C group of the Laboratory for Computer Science at MIT. Since that time, two organizational changes have taken place. LCS merged with the Artificial Intelligence Laboratory at MIT to form a new combined Computer Science and Artificial Intelligence Laboratory, CSAIL (say "sea-sail"). Furthermore, the need was felt to distinguish between the standards-level work done by W3C and the advanced research work. A new group was therefore created, the Decentralized Information Group (DIG), to investigate and

build systems which have web-like structures in general, including the WWW, the Semantic Web, and both technical and social aspects of decentralized systems one might envisage in the future.

9.0 References

aboutMsg:

Dan Connolly, *Convert message metadata to RDF/XML* (schema), 2002. http://www.w3.org/2000/04/maillog2rdf/aboutMsg.py

Algae:

Eric Prud'hommeaux, *Algae RDF Query Language*, May 2004. http://www.w3.org/2004/05/06-Algae/

Amaya:

Irène Vatton, Laurent Carcone, Vincent Quint, *Welcome to Amaya, W3C's Editor/Browser*, 1996. http://www.w3.org/Amaya/

Annotea:

Marja-Riitta Koivunen, *Annotea Project*, February 2001. http://www.w3.org/2001/Annotea/

BAA:

Dr. Jim Hendler (Technical POC):, *The Broad Area Announcement BAA 00-07*; February 4, 2000. [No longer served by DARPA] http://xml.coverpages.org/daml-pipBAA0007.html

cityLookup:

Dan Connolly, *Schema for City Lookup*, 2003. http://www.w3.org/2000/10/swap/pim/cityLookup.n3

CS AKTive:

Nigel Shadbolt and monica schraefel, *CS AKTive Space – A Semantic Web for Research*, April 2003. http://www.aktors.org/akt/events/town2/ppt/11-CSAKtive-Space_files/v3_document.htm

D2RQ:

Chris Bizer, Richard Cyganiak, Jörg Garbers, Oliver Maresch, *D2RQ V0.4 - Treating Non-RDF Databases as Virtual RDF Graphs*, June 2004. http://www.wiwiss.fu-berlin.de/suhl/bizer/d2rq/

DBView:

Dan Connolly, *dbview -- view an SQL DB thru RDF glasses* (schema), 2002. http://www.w3.org/2000/10/swap/dbork/dbview.py

doc.n3:

Tim Berners-Lee, *Documentation control vocabulary* (schema), 2001. http://www.w3.org/2000/10/swap/pim/doc.n3

fink:

Tim Berners-Lee, *Fink to N3* (schema), 2002. http://www.w3.org/2000/10/swap/util/fink2n3.py

Fresnel:

Chris Bizer, Stephen Garland, David Huynh, David Karger, Ryan Lee, Stefano Mazzocchi, Emmanuel Pietriga, Dennis Quan, and Karun Bakshi, *Fresnel - Display Vocabulary for RDF*, October 2004. http://www.w3.org/2005/04/fresnel-info/

fromGarmin:

Tim Berners-Lee, *Download GPS data from serial link to an RDF/N3 file* (schema) 2004. http://www.w3.org/2000/10/swap/pim/fromGarmin.py

fromIcal:

Dan Connolly, *Interpret iCalendar data as RDF* (schema) 2004. http://www.w3.org/2002/12/cal/fromIcal.py

glean:

Dan Connolly, *Glean hCal* (xsl document), August 2005. http://www.w3.org/2002/12/cal/glean-hcal.xsl

GraphViz:

John Ellson et al, Graphviz - Graph Visualization Software, http://www.graphviz.org

Haystack:

David R. Karger (Principal Investigator), Dr. Stephen J. Garland (Research Staff), *Haystack Project,* June 2003. http://haystack.csail.mit.edu/

hCal:

Tantek Çelik and Brian Suda, *hCalendar*, 2004. http://microformats.org/wiki/hcalendar

Ical2:

Dan Connolly, *something.ics >something.rdf* (schema), 2003. http://www.w3.org/2002/12/cal/ical2rdf.pl

IRC:

Jarkko Oikarinen (creator), *Information about IRC: Technical Documents*, 1988. http://www.irc.org/techie.html

IsaViz:

Emmanuel Pietriga, *IsaViz: A Visual Authoring Tool for RDF*, November 2004. http://www.w3.org/2001/11/IsaViz/

jhead:

Tim Berners-Lee, *Program to pull the information out of various types of EXIF digital camera files and show it in a reasonably consistent way* (schema), 2003. http://www.w3.org/2000/10/swap/pim/jhead/jhead.c

Longwell:

Mark Butler, David Franoi Huynh, Ryan Lee, Stefanto Mazzocchi, *Longwell2*, 2004. http://simile.mit.edu/longwell/

lookout:

Tim Berners-Lee, *An attempt to se how one can get into MS Outlook from Python* (schema), 2001. http://www.w3.org/2000/10/swap/pim/lookout.py

make:

Tim Berners-Lee, *Import notation3 (schema)*, 2002.
http://www.w3.org/2000/10/swap/util/make2n3.py

Map Photos:

Tim Berners-Lee, Making a map of the photos on 2004.
http://www.w3.org/2004/lambda/Documents/2004/02/18/Overview.html

mid.proxy:

Dan Connolly, *IMAP<->HTTP proxy service* (schema), 2001.
http://www.w3.org/2000/04/maillog2rdf/mid_proxy.py

MySQL:

David Axmark, Allan Larsson, and Michael "Monty" Widenius, *MySQL software project*,
http://www.mysql.com/

N3 home:

Tim Berners-Lee, editor, *Notation 3:* a *readable language for data on the Web,* 1998.
http://www.w3.org/DesignIssues/Notation3.html

N3 logic:

Tim Berners-Lee, *Notation 3 Logic: An RDF language for the Semantic Web*, August 2005.
http://www.w3.org/DesignIssues/N3Logic

N3 Primer:

Tim Berners-Lee, *Primer: Getting into RDF & Semantic Web using N3*, December 2, 2002.
http://www.w3.org/2000/10/swap/Primer.html

N3 Resources:

Tim Berners-Lee, editor, *Notation 3 Resources: A readable language for the Semantic Web*,
1998. http://www.w3.org/DesignIssues/N3Resources

N3 Tutorial:

Tim Berners-Lee, Dan Connolly, Sandro Hawke, *Semantic Web Tutorial Using N3*, February 6,
2003. http://www.w3.org/2000/10/swap/doc/

NTriples:

Dave Beckett and Art Barstow, *N-Triples: W3C RDF Core WG Internal Working Draft*, July 16,
2002. http://www.w3.org/2001/sw/RDFCore/ntriples/

OFX:

Microsoft, Intuit and CheckFree, *Open Financial Exchange Standard*, 1997.
http://www.ofx.net/ofx/default.asp

ONT:

Sandro Hawke, *Ontaria, Easy Access to the Semantic Web*, April 26, 2005.
http://www.w3.org/2004/ontaria/

org.n3:

Tim Berners-Lee, *Vocabulary for describing the structure of W3C and other organizations which use similar concepts* (schema) 2001. http://www.w3.org/2001/04/roadmap/org.n3

PalmAgent:

Dan Connolly, *PalmAgent -- synchronizing data in my palm with the rest of the Web* (schema), 2001. http://dev.w3.org/cvsweb/2001/palmagent/

PAW:

Dan Connolly (for the PAW Team), *Policy Aware Web*, February 2005. http://www.policyawareweb.org/

Piggy Bank:

David François Huynh, Stefano Mazzocchi, Ryan Lee, *Piggy Bank*, 2004-2006. http://simile.mit.edu/piggy-bank/

PList:

Apple, *Property List Utility*, August 30, 2002. http://developer.apple.com/documentation/Darwin/Reference/ManPages/man1/plutil.1.html

plist2:

Dan Connolly, *PList to RDF* (xsl schema), 2003. http://www.w3.org/2000/10/swap/util/plist2rdf.xsl

Proposal:

Tim Berners-Lee, David R. Karger, Lynn Andrea Stein, Ralph R. Swick, Daniel J. Weitzner, *Semantic Web Development Technical Proposal*, February 4, 2000, http://www.w3.org/2000/01/sw/DevelopmentProposal

Pychinko:

Bijan Parsia, Yarden Katz, and Kendall Clark, *Pychinko: Rete-based RDF friendly rule engine*, January 17, 2005. http://www.mindswap.org/~katz/pychinko/

Python:

Guido van Rossum, Python Software Foundation, 1990, The Python Programming Language Project website. http://www.python.org/

qfx2:

Tim Berners-Lee, *OFX format to N3* (schema), 2004. http://www.w3.org/2000/10/swap/pim/qfx2n3.sed

QIF:

Intuit, *Defining Quicken Interchange Format (QIF) files*, 1997 http://web.intuit.com/support/quicken/2002/win/1177.html

qif2:

Tim Berners-Lee, *QIF interchange format to N3* (schema), 2002. http://www.w3.org/2000/10/swap/pim/qif2n3.py

RDF Cal:

Dan Connolly and Libby Miller, *RDF Calendar - an application of the Resource Description Framework to iCalendar Data*, W3C Interest Group Note, September 20, 2005 http://www.w3.org/TR/rdfcal/

RFC 2445:

F. Dawson, D. Stenerson, *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*, November 1998. http://www.w3.org/2002/12/cal/rfc2445

Roadmap:

Tim Berners-Lee, *Roadmap diagrams*, 2002. http://www.w3.org/2001/04/roadmap/

Rules WS:

Sandro Hawke, *W3C Workshop on Rule Languages for Interoperability*, January 2006, http://www.w3.org/2004/12/rules-ws/

SPARQL:

Eric Prud'hommeaux, Andy Seaborne, editors, *SPARQL Query Language for RDF: W3C Candidate Recommendation*, 6 April 2006. http://www.w3.org/TR/rdf-sparql-query/

SWalker:

Sandro Hawke, *SemWalker*, May 28, 2002. http://www.w3.org/2002/05/semwalker/

swap/contact:

Tim Berners-Lee, *Contact: Utility concepts for everyday life* (schema), 2001. http://www.w3.org/2000/10/swap/pim/contact.n3

Tabulator:

Jim Ley, Tim Berners-Lee, Ruth Dhanaraj, *Tabulator*, 2005. http://dig.csail.mit.edu/2005/ajar/ajaw/tab

TAMI:

Danny Weitzner and Ralph Swick (for DIG),*Transparent Accountable Datamining Initiative*, 2005. http://dig.csail.mit.edu/TAMI/

to.Ical:

Dan Connolly and Tim Berners-Lee, *Convert RDF to iCalendar syntax (schema)*, 2003. http://www.w3.org/2000/10/swap/pim/toIcal.py

TravItin:

Dan Connolly, *Grok itineraries from Navigant; i.e. turn them into RDF/n3, using dublin core and cyc vocabulary (schema)*, 2002. http://www.w3.org/2000/10/swap/pim/grokTravItin.pl

Turtle:

Dave Beckett, *Turtle - Terse RDF Triple Language*, 2002, http://www.dajobe.org/2004/01/turtle/

vcard:

Tim Berners-Lee, *A few rules to extract minimal contact info from RDF MSOutlook vocab and generate VCard files* (schema) 2004. http://www.w3.org/2000/10/swap/pim/mso2vcard.n3

W3C Glance:
Eric Miller, Ryan Lee, *About At A Glance*, March 2003.
http://www.w3.org/2001/10/glance/doc/about

W3C Process:
Ian Jacobs (editor), *World Wide Web Consortium Process Document*, 1996.
http://www.w3.org/2005/10/Process-20051014/

W3C TR:
Tim Berners-Lee, *W3C Technical Reports and Publications*, 1996 http://www.w3.org/TR/

Zakim:
Alan Kotok and Ralph Swick, *The Zakim IRC Teleconference Agent*, 2002.
http://www.w3.org/2001/12/zakim-irc-bot.html

10.0 Appendices

10.1 Appendix A

N3 grammar CFG
http://inamidst.com/n3p/grammar/n3.n3

```
# Notation3 in Notation3

# Context Free Grammar without tokenization
#
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix cfg: <http://www.w3.org/2000/10/swap/grammar/bnf#>.
@prefix rul: <http://www.w3.org/2000/10/swap/grammar/bnf-rules#>.
@prefix : <http://www.w3.org/2000/10/swap/grammar/n3#>.
@prefix n3: <http://www.w3.org/2000/10/swap/grammar/n3#>.
@prefix list: <http://www.w3.org/2000/10/swap/list#>.
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@keywords a, is, of.

# Issues:
# - string token regexp not right  FIXED
# - tokenizing rules in general: whitespace are not defined in n3.n3
#   and it would be nice for the *entire* syntax description to be in RDF.
# - encoding really needs specifying
# - @keywords affects tokenizing
# - Use of dot for !
# - comments (tokenizer deals with)
# - We assume ASCII, in fact should use not notNameChars for i18n

# tokenizing:
# Absorb anything until end of regexp, then stil white space
#  period followed IMMEDIATELY by an opener or name char is taken as "!".
#  Except after a "." used instead of in those circumstances,
#          ws may be inserted between tokens.
#  WS MUST be inserted between tokens where ambiguity would arise.
#  (possible ending characters of one and beginning characters overlap)
#
<> cfg:syntaxFor [ cfg:internetMediaType
                   <http://www.w3.org/2003/mediatypes#application/n3>].
# <> rdfsem:semanticsFor ""   .....
# _____________________________________________________________________
#
# The N3 Full Grammar

language a cfg:Language;
         cfg:document    document;
```

```
	cfg:whiteSpace  "@@@@@".
document a rul:Used;
	cfg:mustBeOneSequence(
				(
#				[ cfg:zeroOrMore declaration ]
#				[ cfg:zeroOrMore universal ]
#				[ cfg:zeroOrMore existential ]
				statements_optional
				cfg:eof
			)
	).
statements_optional cfg:mustBeOneSequence (() ( statement "." statements_optional ) ).

# Formula does NOT need period on last statement

formulacontent cfg:mustBeOneSequence (
			( )
			(
#			 [ cfg:zeroOrMore declaration ]
#			 [ cfg:zeroOrMore  universal ]
#			 [ cfg:zeroOrMore existential ]
			statementlist
	)).

statementlist cfg:mustBeOneSequence (
			( )
			( statement statementtail )
	).
statementtail cfg:mustBeOneSequence (
			( )
			( "." statementlist )
	).

statement  cfg:mustBeOneSequence (
			(declaration)
			(universal)
			(existential)
			(simpleStatement)
	).
universal cfg:mustBeOneSequence (
			(
				"@forAll"
				[ cfg:commaSeparatedListOf symbol ]
			)).

existential cfg:mustBeOneSequence(
			(	 "@forSome"
				 [ cfg:commaSeparatedListOf symbol ]
			)).

declaration cfg:mustBeOneSequence(
```

```
                ( "@prefix" qname explicituri  )
                ( "@keywords" [ cfg:commaSeparatedListOf barename ] )
        ).
simpleStatement cfg:mustBeOneSequence(( subject propertylist )).

propertylist cfg:mustBeOneSequence (
                ( )
                ( verb  object objecttail propertylisttail )
        ).
propertylisttail cfg:mustBeOneSequence (
                ( )
                ( ";" propertylist )
        ).
objecttail cfg:mustBeOneSequence (
                ( )
                ( ","   object objecttail )
        ).
verb cfg:mustBeOneSequence (
                ( path )
                ( "@has" path )
                ( "@is" path "@of" )
                ( "@a" )
                ( "=" )
                ( "=>" )
                ( "<=" )
        ).
# prop cfg:mustBeOneSequence ((node)).

subject cfg:mustBeOneSequence ((path)).

object cfg:mustBeOneSequence ((path)).

path cfg:mustBeOneSequence(
                ( node pathtail )
        ).
pathtail cfg:mustBeOneSequence(
                (  )
                ( "!" path )
                ( "^" path )
        ).
node cfg:mustBeOneSequence (
                ( symbol )
                ( "{" formulacontent "}" )
                ( variable )
                ( numericliteral )
                ( literal )
                ( "[" propertylist "]"  )
                (  "("  pathlist ")"  )
                ( "@this" )          #  Deprocated.  Was allowed for this log:forAll x
).
pathlist cfg:mustBeOneSequence (() (path pathlist)).
```

```
symbol cfg:mustBeOneSequence (
                (explicituri)
                (qname)
        ).
literal cfg:mustBeOneSequence(( string dtlang)).

dtlang cfg:mustBeOneSequence(  ()  ("@" langcode)  ("^^" symbol)).

#_____________________________________________________________________
#
#   TERMINALS

numericliteral  cfg:matches     """[-+]?[0-9]+(\\.[0-9]+)?(e[-+]?[0-9]+)?""";
                cfg:canStartWith        "0", "-", "+".

explicituri     cfg:matches     "<[^>]*>";
                cfg:canStartWith        "<".

qname           cfg:matches     "(([a-zA-Z_][a-zA-Z0-9_]*)?:)?([a-zA-Z_][a-zA-Z0-9_]*)?";
                cfg:canStartWith        "a", "_", ":".  # @@ etc unicode

barename        cfg:matches     "[a-zA-Z_][a-zA-Z0-9_]*";  # subset of qname
                cfg:canStartWith        "a", "_".  # @@ etc

variable        cfg:matches     "\\?[a-zA-Z_][a-zA-Z0-9_]*";  # ? barename
                cfg:canStartWith        "?".  #

# Maybe dtlang should just be part of string regexp?
# Whitespace is not allowed

# was: "[a-zA-Z][a-zA-Z0-9]*(-[a-zA-Z0-9]+)?";
langcode        cfg:matches     "[a-z]+(-[a-z0-9]+)*"; # http://www.w3.org/TR/rdf-testcases/#language
                cfg:canStartWith        "a".

#           raw regexp single quoted would be   "([^"]|(\\"))*"
# See:
#       $ PYTHONPATH=$SWAP python
#       >>> import tokenize
#       >>> import notation3
#       >>> print notation3.stringToN3(tokenize.Double3)
#       "[^\"\\\\]*(?:(?:\\\\.|\"(?!\"\"))[^\"\\\\]*)*\"\"\""
#       >>> print notation3.stringToN3(tokenize.Double)
#       "[^\"\\\\]*(?:\\\\.[^\"\\\\]*)*\""
# After that we have to prefix with one or three opening \"  which
# the python regexp doesn't have them.
#
# string3               cfg:matches             "\"\"\"[^\"\\\\]*(?:(?:\\\\.|\"(?!\"\"))[^\"\\\\]*)*\"\"\"".
# string1               cfg:matches             "\"[^\"\\\\]*(?:\\\\.[^\"\\\\]*)*\"".
```

```
string          cfg:matches
        "(\"\"\"[^\"\\\\]*(?:(?:\\\\.|\"(?!\"\"))[^\"\\\\]*)*\"\"\")|(\"[^\"\\\\]*(?:\\\\.[^\"\\\\]*)*\")";
                cfg:canStartWith        "\"".
#_____________________________________________________________

#  Axioms reducing the shortcut CFG terms to cfg:musBeOneSequence.

{ ?x cfg:zeroOrMore ?y } => {?x cfg:mustBeOneSequence ( () (?y ?x) ) }.

{ ?x cfg:commaSeparatedPeriodTerminatedListOf ?y } =>
{
        ?x cfg:mustBeOneSequence (
                ( "." )
                ( ?y [cfg:CSLTail ?y]  )
        )
}.
{ ?x cfg:CSLTail ?y } =>
{
        ?x cfg:mustBeOneSequence (
                ( "." )
                ( "," ?y ?x )
        )
}.
# Without the period
{ ?x cfg:commaSeparatedListOf ?y } =>
{
        ?x cfg:mustBeOneSequence (
                (  )
                ( ?y [cfg:CSLTail2 ?y]  )
        )
}.
{ ?x cfg:CSLTail2 ?y } =>
{
        ?x cfg:mustBeOneSequence (
                ( )
                ( "," ?y ?x )
        )
}.
#  labelling of things which do not have explicit URIs:

{ ?x cfg:zeroOrMore [ cfg:label ?y].
        ( ?y "_s" ) string:concatenation ?str } => { ?x cfg:label ?str }.

{ ?x cfg:commaSeparatedPeriodTerminatedListOf [ cfg:label ?y].
        ( ?y "_csl" ) string:concatenation ?str } => { ?x cfg:label ?str }.

{ ?x cfg:CSLTail [ cfg:label ?y].
        ( ?y "_necsl" ) string:concatenation ?str } => { ?x cfg:label ?str }
```

Appendix B
N3logic

http://www.w3.org/DesignIssues/N3Logic

Tim Berners-Lee, August 2005
Status: An early draft of a semi-formal semantics of the N3 logical properties.

An RDF language for the Semantic Web

Notation 3 Logic

This article gives operational semantics for Notation3 (N3) and some RDF properties for expressing logic. These properties, together with N3's extensions of RDF to include variables and nested graphs, allow N3 to be used to express rules in a web environment.

This is an informal semantics in that it should be understandable by a human being but is not a machine readable formal semantics. This document is aimed at a logician wanting a reference by which to compare N3 Logic with other languages, and at the engineer coding an implementation of N3 Logic and wanting to check the detailed semantics.

These properties are not part of the N3 language, but are properties which allow N3 to be used to express rules, and rules which talk about the provenance of information, contents of documents on the Web, and so on. Just as OWL is expressed in RDF by defining properties, rules, queries, differences, and so on can be expressed in RDF with the N3 extension to formulae.

The log: namespace has functions which have built-in meaning for cwm and other software.

See also:

- The schema for the log: namespace
- A vocabulary for expressing differences between RDF graphs
- a formal design for RDF/N3 context/scopes
  Dan Connolly to www-rdf-logic, Thu, Sep 06 2001

The prefix log: is used below as shorthand for the namespace <http://www.w3.org/2000/10/log#>. See the schema for a summary.

Motivation

The motivation of the logic was to be useful as a tool in an open web environment. The Web contains many sources of information, with different characteristics and relationships to any given reader. Whereas a closed system may be built based on a single knowledge base of believed facts, an open web-based system exists in an unbounded sea of interconnected information resources. This requires that an agent be aware of the provenance of information, and responsible for its disposition. The language for use in this environment typically requires the ability to express what document or message said what, so the ability to quote subgraphs and match them against variable graphs is essential. This quotation and reference, with its inevitable possibility of direct or indirect self-reference, if added directly to first order logic presents problems such as paradox traps. To avoid this, N3 logic has deliberately been kept to limited expressive power: it currently contains no general first order negation. Negated forms of many of the built-in functions are available, however.

A goal is that information, such as, but not limited to rules, which requires greater expressive power than the RDF graph, should be sharable in the same way as RDF can be shared. This means that one person should be able to express knowledge in N3 for a certain purpose, and later someone else independently reuse that knowledge for a different unforeseen purpose. As the context of the later use is unknown, this prevents us from making implicit closed assumptions about the total set of knowledge in the system as a whole.

Furthermore, we require that other users of N3 in the Web be able to express new knowledge without affecting systems we have already built. This means that N3 must be fundamentally monotonic: the addition of new information from elsewhere, while it might cause an inconsistency by contradicting the old information (which would have to be resolved before the combined system is used), cannot silently change the meaning of the original knowledge.

The non-monotonicity of many existing systems follows from a form of negation known as failure, in which a sentence is deemed false if it not held within (or derivable from) the *current knowledge base*. It is this concept of current knowledge base, which is a variable quantity, and the ability to indirectly make reference to it which causes the non-monotonicity. In N3Logic, while a current knowledge base is a fine concept, there is no ability to make reference to it implicitly in the negative. The negation provided is the ability only for a specific given document (or, essentially, some abstract formula) to objectively determine whether or not it holds, or allows one to derive, a given fact. This has been called *Scoped Negation As Failure* (SNAF).

Formal syntax

The syntax of N3 is defined by the context-free grammar. This is available in machine-readable form in Notation3 and RDF/XML.

The top-level production for an N3 document is <http://www.w3.org/2000/10/swap/grammar/n3#document>.

In the semantics below we will consider these productions using notation as follows.

Production	N3 syntax examples	notation below for instances
symbol	**<foo#bar> <http://example.com/>**	c d e f
variable	Any symbol quantified by @forAll or @forSome in the same or an outer formula.	*x y z*
formula	**{ ... }** or an entire document	F G H K
set of universal variables of F	**@forAll :x, :y.**	uvF
set of existential variables of F	**@forSome :z, :w.**	evF
set of statements of F		stF
statement	**<#myCar> <#color> "green".**	F*i* or {s p o}
string	**"hello world"**	s
integer	**34**	i
list	(1 2 ?x <a>)	L M
Element i of list L		L*i*

length of list		|L|
expression	see grammar	n m
Set*	{$ 1, 2, <a> $}	S T

*The set syntax and semantics are not part of the current Notation3 language, but are under consideration.

Semantics

Note. The Semantics of a generic RDF statement are not defined here. The extensibility of RDF is deliberately such that a document may draw upon predicates from many sources. The statement {n c m} expresses that the relationship denoted by c holds between the things denoted by n and m. The meaning of the statement {n c m} in general is defined by any specification for c. The Architecture of the WWW specifies informally how the curious can discover information about the relation. It discusses how the architecture and management of the WWW is such that a given social entity has jurisdiction over certain symbols (though for example domain name ownership). This philosophy and architecture is not discussed further here. Here, though, we do define the semantics of certain specific predicates which allow the expression of the language. In analyzing the language, the reader is invited to consider statements of unknown meaning ground facts. N3Logic defines the semantics of certain properties. Clearly a system which recognizes further logical predicates, beyond those defined here, whose meaning introduces greater logical expressiveness would change the properties of the logic.

Simplifications

N3 has a number of types of shortcut syntax and syntactic sugar. For simplicity, in this article we consider a language simpler than the full N3 syntax referenced above though just as expressive, in that we ignore most syntactic sugar. The following simplifications are made:

We ignore syntactic sugar of comma and semicolon as shorthand notations. That is, we consider a simpler language in which any such syntax has been expanded out. Loosely:

A sentence of the form	becomes two sentences
subject *stuff* ; *morestuff* .	subject *stuff* . subject *morestuff* .
subject predicate *stuff* , object .	subject predicate *stuff* subject predicateobject .

For those familiar with N3, the other simplifications in the language considered here are as follows:

- prefixes have been expanded and all qualified names replaced with symbols using full URIs between angle brackets;
- the path syntax which uses "!" and "^" is assumed to be expanded into its equivalent blank node form;
- the "is ... of " backwards construction has been replaced by the equivalent forwards direction syntax;
- the "=" syntax is not used as shorthand for owl:sameAs. In fact, we use = here in the text for value equality;
- @keywords is not used;
- the @a shorthand for rdf:type is replaced with a direct use of the full URI symbol for rdf:type;
- all ?x forms are replaced with explicit universal quantification in the enclosing parent of the current formula.

Notation3 has explicitly quantified existential variables as well as blank nodes. The description below does not mention blank nodes, although they are very close in semantics to existentially quantified variables. We consider for now a simpler language in which blank nodes have been replaced by explicitly named variables existentially quantified in the same formula.

We have only included strings and integers, rather than the whole set of RDF types and user-defined types.

These simplifications will not deter us from using N3 shorthand in examples where it makes them more readable, so the reader is assumed familiar with them.

Defining N3 Entailment

The RDF specification defines a very weak form of entailment, known as RDF entailment or simple entailment. Here we define the equivalent and very simple N3-entailment. This does not provide us with useful powers of inference: it is almost textual inclusion, but just has conjunction elimination (statement removal) , universal elimination, existential introduction and variable renaming. Most of this is quite traditional. The only thing that distinguishes N3 Logic from typical logics is the Formula, which allows N3 sentences to make statements about N3 sentences. The following details are included for completeness and may be skipped.

Substitution

Substitution is defined to recursively apply inside compound terms, as is usual. Note that substitution descends into compound terms, while substitution of owl:sameAs, discussed later, does not.

We define a substitution operator $\sigma_{x/m}$ which replaces occurrences of the variable *x*. with the expression m. For compound terms, substitution of a compound term (list, formula or set) is performed by performing substitution of each component recursively.

Abbreviating the substitution $\sigma_{x/m}$ as σ , we define the substitution operator as usual:

σ*x* = m (*x* is replaced by m)
σ*y* = *y* (y not equal to x)
σa = a (symbols and literals are unchanged)
σi = i
σs = s
σ (a b ... c) = (σa σb ... σc) (substitution goes into compound terms)
σ{$ a, b, ... c $} = {$ σa, σb, ... σc $}
uv σF = σ uvF
ev σF = σ evF
st σF = σ stF

In general a substitution operator is the sequential application of single substitutions:

$\sigma=\sigma_{x1/m1}\sigma_{x2/m2}\sigma_{x2/m2}$ ... $\sigma_{xn/mn}$

Value equality

Value equality between terms is defined in an ordinary way, compatible with RDF.

For concepts which exist in RDF, we use RDF equality. This is RDF node equality. These atomic concepts have a simple form of equality.

For lists, equality is defined as a pairwise matching.

For sets, equality is defined as a mapping between equal terms existing in each direction.

For formulae, equality F = G is defined as a substitution σ existing mapping variables to variables. (Note that as here RDF Blank Nodes are considered as existential variables, the substitution will map b-nodes to b-nodes.)

The table below is a summary for completeness.

Production	Equality
symbol	uri is equal unicode string
variable	variable name is equal unicode string
formula	F = G iff |stF| = |stG| and there is some substitution œÉ such that(,□*i* ., □*j* . σ*Fi* = σG*j*.)
statement	Subjects are equal, predicates are equal, and objects are equal
string	equal unicode string
integer	equal integer
list L = M	|L| = |M| & (,□*i* . L*i* = M*i*)
set S = T	(,□*i* ., □*j* . S*i* = T*j*.) & (,□*i* ., □*j* . S*i* = T*j*.)
formula F = G	,□σ . σ F = σ G
unicode string	Unicode strings should be in canonical form. They are equal if the corresponding characters have numerically equal code points.

Conjunction

N3, like RDF, has an implied conjunction, with its normal properties, between the statements of a formula.

The semantics of a formula which has no quantifiers (@forAll or @forSome) is the conjunction of the semantics of the statements of which it is composed.

We define the conjunction elimination operator ce(i) as removing the statement F*i* from formula F. By the conventional semantics of conjunction, the ce(i) operator is truth-preserving. If you take a formula and remove a statement from it, it is still true.
CE: From F follows ce(i) F

Existential quantification

Existential quantifiers and Universal quantifiers have the usual qualities
Any formula, including the *root formula,* which matches the "document" production of the grammar, may have a set of existential variables indicated by a `@forSome` declaration. This indicates where the formula is considered true, it is true for at least one substitution mapping the existential variables onto non-variables.

As usual, we define a truth-preserving Existential Introduction operator on formulae, that of introducing an existentially quantified variable in place of any term. The operation ei(x, n) is defined as

1. Creation of a new variable *x* which occurs nowhere else
2. The application $\sigma_{x/n}$ to F
3. The addition of *x* to evF.

EI: From F follows ei(x,n) F for any *x* not occurring anywhere else

Universal quantification

Any formula, including the root formula, may have a set of universal variables. These are indicated by @forall declarations. The scope of the **@forAll** is outside the scope of any @forSome.

If both universal and existential quantification are specified for the same context, then the scope of the universal quantification is outside the scope of the existentials:

{ @forAll <#h>. @forSome <#g>. <#g> <#loves> <#h> }.

means

□<#h> (, □ <#g> ((<#g> <#loves> <#h>))

The semantics of @forAll is that for any substitution œÉ = subst(*x*, n) where x member of uvF, if F is true then œÉF is also true. Any @forAll declaration may also be removed, preserving truth. Combining these, we define a truth-preserving operation ue(x, n) such that ue(x, n) F is formed by

1. Removal of x from evF
2. Application of subst(x, n)

We have the axiom of universal elimination

UE: From F follows ue(x, n) F for all x in evF

As the actual variable used in a formula is quite irrelevant to its semantics, the operation of replacing that variable with another one not used elsewhere within the formula is truth-preserving.

Variable renaming

We define the operation of variable renaming vr(*x*,*y*) on F when x is a member of uvF or is a member of evF.

VR: From F follows vr(*x*, *y*) F where *x* is in uvF or evF and *y* does not occur in F

Occurrence in F is defined recursively in the same way as substitution: *x* occurs in F iff $\sigma_{x/n}$F is not equal to F for arbitrary n.

Union of formulae

The union H = F ∗ G of two formulae F and G is formed, as usual, as follows.

A variable renaming operator is applied to G such that the resulting formula G' has no variables that occur unquantified, differently quantified, nor existentially quantified in F, and vice-versa. (F and G' may share universal variables).

F ∗ G is then defined by:

st(F ∗ G) = stF ∗ st G' ; ev(F ∗ G) = evF ∗ evG' ; uv(F ∗ G) = uvF ∗ uv G'

N3 entailment

The operators conjunction elimination, existential elimination, universal introduction and variable renaming are truth preserving. We define an N3 entailment operator (τ) as any operator which is the successive application of any sequence (possibly empty) of such operators. We say a formula F n3-entails a formula τ F. By a combination of SE, EI, UE and VR, τ F logically follows from F.

Note. RDF Graph is a subclass of N3 formula. If F and G are RDF graphs, only CI and EI apply and n3-entailment reduces to simple entailment from RDF Semantics. (@@check for any RDF weirdness)

We have now defined this simple form of N3-entailment, which amounts to little more than textual inclusion in one expression of a subset of another. We have not defined the normal collection of implication, disjunction and negation for first order logic, as N3logic provides for first order negation. We have, in the process, defined a substitution operation which we can now use to define implication, which allows us to express rules.

Logic properties and built-in functions

We now define the semantics of N3 statements whose predicate is one of a small set of logic properties. These are statements whose truth can be established by performing calculations, or by accessing the Web.

One of our objectives was to make it possible to make statements about and query other statements, such as the contents of data in information resources on the web. We have, in formulae, the ability to represent such sets of statements. Now, to allow statements about them, we take some of the relationships we have defined and give them URIs, so that these statements and queries can be written in N3.

While the properties we introduced can be used simply as ground facts in a database, is very useful to take advantage of the fact that they can be calculated. In some cases, the truth or falsehood of a binary relation can be calculated; in others, the relationship is a function, so one argument (subject or object of the statement) can be calculated from the other.

We now show how such properties are defined and give examples of how an inference system can use them. A motivation here is to do for logical information what RDF did for data: to provide a common data model and common syntax, so that extensions of the language are made simply by defining new terms in an ontology. Declarative programming languages like scheme [@@] of course do this. However, they differ in their choice of pairs rather than the RDF binary relational model for data, and lack the use of universal identifiers as symbols. The goal with N3 was to make a minimal extension to the RDF data model, so that the same language could be used for logic and data, which, in practice, are mixed as a colloidal solution in many real applications.

Calculated entailment

We introduce also a set of properties whose truth may be evaluated directly by machine. We call these "built-in" functions. The implementation as built-in functions is not, in general, required for any implementation of the N3 language, as they can always soundly be treated as ground facts. However, their usefulness derives from their implementation. We say that, for example { 1 math:negation -1 } is entailed by calculation. Like other RDF properties, the set is designed to be extensible, as others can use URIs for new functions. A much larger set of such properties is described for example in the cwm built-ins list, and the semantics of those are not described here.

When the truth of a statement can be deduced because its predicate is a built-in function, we call the derivation of the statement from no other evidence *calculated entailment.*

We now define a small set of such properties which provide the power of N3 logic for inference on the Web.

log:includes

If a formula G n3-entails another formula F, this is expressed in N3 logic as

F log:includes G.

Note. In deference to the fact that RDF treats lists not as terms, but as things constructed from first and rest pairs, we can view formulae which include lists as including rdf:first and rdf:rest statements. The effect on inclusion is that two other entailment operations are added: the addition of any statement of the form ***L rdf:first n,*** *where n is the first element of L, or L rdf:rest K where K is list forming the remaining non-first elements of L. This is not essential to a further understanding of the logic, nor to the operation of a system which does not contain any explicit mention of the terms rdf:first or rdf:rest.*

For the discussion of n3-entailment, clearly:

From F and F log:includes G logically follows G

This can be calculated because it is a mathematical operation on two compound terms. It is typically used in a query to test the contents of a formula. Below we will show how it can be used in the antecedent of a rule.

log:notIncludes

We write of formulae F and G: F log:notIncludes G if it is **not** the case that G n3-entails F.

As a form of negation, log:notincludes is completely monotonic. It can be evaluated by a mathematical calculation on the value of the two terms: no other knowledge gained can influence the result. This is the *scoped negation as failure* mentioned in the introduction. This is not a non-monotonic negation as failure.

Note on computation: To a certain whether G n3-entails F in the worst case involves checking for all possible n3-entailment transformations which are combinations of the variables which occur in G. This operation may be tedious; it is strictly graph isomorphism complete. However, the use of symbols rather than variables for a good proportion of nodes makes it much more tractable for practical graphs. The ethos that it is a good idea to give name things with URIs (symbols in N3) is a basic meme of Web architecture [AWWW]. It has direct practical application in the calculation of n3-entailment, as comparison of graphs whose nodes are labeled is much faster (of order n log (n)))

log:implies

The log:implies property relates two formulae, expressing implication. The shorthand notation for log:implies is => . A statement using log:implies, unlike log:includes, cannot be calculated. It is not a built-in function, but the predicate which allows the expression of a rule.

The semantics of implication are standard, but we elaborate them now for completeness.

F log:implies G is true if and only if when the formula F is true then also G is true.

MP: From F and F => G follows G

A statement in formula H is of the form F=>G can be considered as rule, in which case the subject F is the premise (antecedent) of the rule, and the object G is the consequent.

Implication is normally used within a formula with universally quantified variables.

For example, universal quantifiers are used with a rule in H as follows. Here H is the formula containing the rules, and K the formula upon which the rules are applied, which we can call the knowledge base.

If F => G is in H, and then for every σ which is a transformation composed of universal eliminations of variables universally quantified in H, then it also follows that σ F => σ G. Therefore, for every σ such that K includes σF, σG follows from K.

In the particular case that H and K are both the knowledge base, or formula believed true at the top level, then

GMP: From F => G and σF follows σG if σ is a transformation composed of universal eliminations of variables universally quantified at the top level.

Filtering

When a knowledge base (formula) contains a lot of information, one way to filter off a subset is to run a set of rules on the knowledge base and take only the new data generated by the rules. This is the filter operation.

When you apply rules to a knowledge base, the *filter result* of rules in H applied to K is the union of all σG for every statement F => G which is in H, for every σ which is a transformation composed of universal eliminations of variables universally quantified in H such that K includes σF.

Repeated application of rules

When rules are added repeatedly into the same knowledge base there is a check to see whether the H already includes σG before adding σG to it and then if it does, skipping it, in order to prevent the unnecessary extra growth of the knowledge base,.

Let the result of rules in H applied to K, ρ_HK, be the union of K with all σG for every statement F => G which is in H, for every σ which is a transformation composed of universal eliminations of variables universally quantified in H, such that K includes σF, and K does not n3-entail σG.

Note. This form of rule allows existentials in the consequent: it is not datalog. It is clearly possible in a forward-chaining reasoner to generate an unbounded set of conclusions with rules of the form (using shorthand)

{ ?x a:Person } => { ?x :mother [a :Person] }.

While this is a trap for the unwary user of a forward-chaining reasoner, it was found to be essential, in general, to be able to generate arbitrary RDF containing blank nodes, for example when translating information from one ontology into another.

Consider the repeated application of rules in H to K, ρi_HK. If there are no existentially quantified variables in the consequents of any of the rules in H, then this is like datalog and there will be some threshold *n* above which no more data is added, and a closure: ρi_HK = ρn_HK for all *i*>*n*. In fact, in many practical applications, even with the datalog constraint removed, there is also a closure. This $\rho^{\infty}{}_H$K is the result of running a forward-chaining reasoner on H and K.

Rule Inference on the knowledge base
In the case in which rules are in the same formula as the data, the single rule operation can be written $\rho_K K$, and the closure under rule application $\rho^{\infty}_K K$

Cwm note: the --rules command line option calculates $\rho_K K$ and the --think calculates $\rho^{\infty}_K K$. The --filter=H calculates the filter result of H on the knowledge base.

Examples
Here a simple rule uses log:implies.

```
@prefix log: .
@keywords.
@forAll x, y, z. {x parent y. y sister z} log:implies {x aunt z}
```

This N3 formula has three universally quantified variables and one statement. The subject of the statement,

{x parent y. y sister z}

is the antecedent of the rule and the object,

{x aunt z}

is the conclusion. Given data

Joe parent Alan.
Alan sister Susie.

a rule engine would conclude

Joe aunt Susie.

As a second example, we use a rule which looks inside a formula:

```
@forAll x, y, z.
{ x wrote y.
  y log:includes {z weather w}.
  x home z
} log:implies {
  Boston weather y
}
```

Here the rule fires when x is bound to a symbol denoting some person who is the author of a formula y, when the formula makes a statement about the weather in (presumably some place) z, and x's home is z. That is, we believe statements about the weather at a place only from people who live there. Given the data

Bob lives Boston.
Bob wrote { Boston weather sunny }.
Alice lives Adelaide.
Alice wrote { Boston weather cold }

a valid inference would be

Boston weather sunny.

log:supports

We say that F log:supports G if there is some sequence of rule inference and/or calculated entailment and/or n3 entailment operators which when applied to F produce G.

log:conclusion

The log:conclusion property expresses the relationship between a formula and its deductive closure under operations of n3-entailment, rule entailment and calculated entailment.

As noticed above, there are circumstances when this will not be finite.

log:conclusion is the transitive closure of log:supports.

log:supports can be written in terms of log:conclusion and log:includes.

{ ?x log:supports ?y } if and only iff { ?x log:conclusion [log:includes ?y]}

However, log:supports may be evaluated in many cases without evaluating log:conclusion: one can determine whether y can be derived from x in many ways, such as backward chaining, without necessarily having to evaluate the (possibly infinite) deductive closure.

Now we have a system which has the capacity to do inference using rules and operate on formulae. However, it operates in a vacuum. In fact, our goal is that the system should operate in the context of the Web.

Involving the Web

We therefore expose the web as a mapping between URIs and the information returned when such a URI is dereferenced, using appropriate protocols. In N3, the information resource is identified by a symbol, which is in fact is its URI. In N3, information is represented in formulae, so we represent the information retrieved as a formula. Not all information on the web is, of course, in N3. However, the architecture we design is that N3 should here be the interlingua. Therefore, from the point of view of this system, the semantics of a document is exactly what can be expressed in N3 - no more and no less.

log:semantics**

c log:semantics F is true iff c is a document whose logical semantics expressed in N3 is the formula F.

The relation between a document and the logical expression which represents its meaning is expressed as N3. The Architectures of the World Wide Web [AWWW] defines algorithms by which a machine can determine representations of document given its symbol (URI). For a representation in N3, this is the formula which corresponds to the *document* production of the grammar. For a representation in RDF/XML, it is the formula, which is the entire graph parsed. For any other languages, it may be calculated in as much a specification exists which defines the equivalent N3 semantics for files in that language.

On the meaning of N3 formula

This is not, of course, the semantics of the document in any absolute sense. It is the semantics expressed in N3. In turn, the full semantics of an N3 formula are grounded, in the definitions of the properties and classes used by the formula. In the HTTP space in which URIs are minted by an authority, definitive information about those definitions may be found by dereferencing the URIs. This information may be in

natural language, in some machine-processable logic, or a mixture. Two patterns are important for the Semantic Web.

One is the grounding of properties and classes by defining them in natural language. Natural language, of course, is not capable of giving an absolute meaning to anything in theory, but, in practice, a well written document carefully written by a group of people achieves a precision of definition which is quite sufficient for the community to be able to exchange data using the terms concerned. The other pattern is the raft-like definition of terms in terms of related neighboring ontologies.

@@@@ A full discussion of the grounding of meaning in a web of such definitions is beyond the scope of this article. Here we define only the operation semantics of a system using N3.

@@@@ Edited up to here

The log:semantics of an N3 document is the formula achieved by parsing representation of the document.
(Cwm note: Cwm knows how to go get a document and parse N3 and RDF/XML
it in order to evaluate this.)

Other languages for web documents may be defined whose N3 semantics are therefore also calculable, and so they could be added in due course.
See for example [GRDDL], [RDF/A], etc

However, for the purpose of the analysis of the language, it is a convenient to consider the Semantic Web simply as a binary 1:1 relation between a subset of symbols and formulae.

For a document in Notation3, log:semantics is the
log:parsedAsN3 of the log:contents of the document.

log:says
log:says is defined by:

F log:says G iff □ H . **F log:semantics H and H log:includes G**

In other words, a document says something if a representation of it in the sense of the Architecture of the World Wide Web [AWWW] N3-entails it.
The semantics of log:says are similar to that of says in [PCA].

Miscellaneous
log:Truth

This is a class of true formulae.

From { F rdf:type log:Truth } follows F

The cwm engine will process rules in the (indirectly command-line specified) formula or any formula which that declares to be a Truth.

The dereifier will output any described formulae which are described as being in the class Truth.

This class is not at all central to the logic.

Working with OWL

@@ Summary

- owl:sameAs considered the same as N3 value equality for data values. Axioms of equality. log:equalTo and log:notEqualTo compared with owl:SameAs. Compare math and string equality, and sparql equality.
- Operating in equality-aware mode.
- No attempt at connecting OWL DL language with the N3 logic.
- Use of functional properties of a datatype conflicting with OWL DL.

Conclusion

The semantics of N3 have been defined, as have some built-in operator properties which add logical inference using rules to the language, and allow rules to define inference which can be drawn from specific web documents on the Web, as a function of other information about those documents.

The language has been found to have some useful practical properties. The separation between the Notation3 extensions to RDF and the logic properties has allowed N3, by itself, to be used in many other applications directly, and with other properties to provide other functionality, such as the expression of patches (updates) [Diff].

The use of log:notIncludes to allow default reasoning without non-monotonic behavior achieves a design goal for distributed rule systems.

**[Footnote: Philosophers may be distracted here into worrying about the meaning of meaning. At least we did not call this function "meaning!" In as much as N3 is used as an interlingua for interoperability for different systems, this for an N3 based system is the meaning expressed by a document. One reviewer was aghast at the definition of semantics as being that of retrieval of a representation, its parsing and assimilation in terms of the local common logical framework. I suspect, however, that the meaning of the paper to the reviewer could be considered quite equivalently the result of the process of retrieval of a representation of the paper, its parsing by the review, and its assimilation in terms of the reviewer's local logical framework: a similar, though perhaps imperfect, process.
Of course, the semantics of many documents are not expressible in logica at all, and many in logic but not in N3. However, we are building a system for which a prime goal is the reading and investigation of machine-readable documents on the Web. We use the URI log:semantics for this function and apologize for any heartache it may cause.]

F = G iff |stF| = |stG| and there is some substitution œÉ such that(‚àÄ*i* .‚àÉ*j* . œÉ*Fi* = œÉG*j*.)

Appendix: Colophon

formatting XHTML 1 with nvu

Appendix: Drafting Notes

yes, discuss notational abbreviation, but not abstract syntax

hmm... are log:includes, log:implies and such predicates? relations? operators? properties?

To do: describe the syntactic sugar transformations formally to close the loop10.3 Appendix C

CWM, a general purpose processor for the Semantic Web

http://www.w3.org/2000/10/swap/doc/cwm.html

Cwm

Cwm (pronounced coom) is a general-purpose data processor for the Semantic Web, somewhat like sed, awk, etc. for text files or XSLT for XML. It is a forward chaining reasoner which can be used for querying, checking, transforming and filtering information. Its core language is RDF, extended to include rules, and it uses RDF/XML or RDF/N3 (see Notation3 Primer) serializations as required.

Cwm is written in python; it is part of SWAP, a Semantic Web Application Platform. It is open source under the W3C software license.

Quick Reference:

- Command line syntax
- Built-in functions
- Installation
- Change log, Plans, Bugs

Discussion -- places to talk about this

public-cwm-announce
: This low-traffic is for announcements about releases of cwm software, and monthly summaries of changes to the bug/RFE list. You might find the cwm-announce RSS feed handy.

public-cwm-bugs
: This is for the announcement and brief discussion/clarification of cwm bugs or Requests For Enhancement (RFE). If responding to an existing bug, only use mailers which send the refernce headers so that the threads on this mail ling list work. For new threads, please make the subject line informative, and use the word "bug" or "RFE" as appopriate. The current plan is to review changes in this monthly and send it to the announce list.

public-cwm-talk
: Discussion by users and/or developers of the use and abuse of project software.

Semantic Web Interest group

- The RDF Interest group discussion list
- #swig irc channel with scratchpad/weblog

Features and Tutorial

The Semantic Web Tutorial Using N3 covers features such as:

- Loading files in RDF/XML and/or N3, generating RDF or N3 files from the result.
  - (The obscureboring parts of RDF/XML syntax, specifically reification and XML Literal parse type, are **not** handled by the main parser).
- Pretty printing data so that anonymous nodes are used creatively to minimize the number of explicit existentials (generated Ids).
- Applying rules written in N3 to the data
- Filtering the data to the result of a particular query

- Generating arbitrary formats (using --strings)
- Using an internal knowledge of functions to resolve them within a query, including:
    - Simple math and string operations
    - Getting and parsing documents from the web
    - Accessing command line arguments and environment variables
    - Cryptography: hashing, generating keys, signing things and checking signatures.

See also: Cwm command line arguments reference.

Other features are in development, and haven't been documented as thoroughly:

- Accessing the web to directly or indirectly resolve a query, including:
    - Getting schemas for terms in the query
    - Using metadata to point to definitive documents
    - Looking up data in local or remote SQL servers

Environment Variables

CWM_RDF_PARSER

rdflib2rdf or sax2rdf (default). Affects the choice of RDF parser module used by cwm.

Security Issues

Be careful when using rules from an untrusted source.

- Rules can read data from the Web, indirectly letting data out by the URIs they use.
- Rules can take up your resources such as processor time and memory.
- Rules can pick data up from within the web you have access to, including confidential files.

Be carfeul even when using cryptography. I am not an expert but a few things to watch are:

- Always think where the weakest link is. It is not always on the net.
- Where do you keep the private key, anyway?
- Beware of all forms of attack, including replay and man in the middle.
- Always sign some random junk as well as the critical data to prevent the reverse engineering of the key.
- Ask a crypto specialist to look over your stuff
- Make the techniques, rules, code. public. Public debugging is valuable. Trying to hide it from attackers by keeping it secret doesn't pay.
- This code is not guaranteed anyway, or made for production use. It is designed for prototyping new semantic web applications. Use at your own risk.

About the name *cwm*

Originally, the name is from from "Closed World Machine" because it processed information in a limited space, cwm does *not* make any assumptions about a closed world. Think of it as defined area but with openings - like a valley.

Related Work

see also Sean Palmer's guide to cwm -- sometimes it is more up to date than this!

Check out other programs which use the same language:

- Euler - a backward-chaining reasoner by Jos de Roo. Euler will tell you whether a give set of facts and rules supports a give conclusion.
- EulerSharp - a C# port of Euler
- cwmclone - a partial clone of cwm by Bijan Parsia to XSB prolog engine - to demonstrate that conventional logica programming tools are efficent and straightforwradly adapted to semantic web work.
- Jena RDF toolkit now accepts N3 as well as RDF/XML (2003/2)
- RDF::Notation3 perl module (submission 10 Oct 2001 by Petr Cimprich)
- Swish - N3 -capable semantic web code Haskell by Graham Kline (2003/6)
- Pychinko is a rete-based cwm clone - should be much faster.

Development

This swap code is open source and available for those that want to play with it, but comes with no warranty.

Using CVS

from the public w3c CVS repository. Check out the whole tree to develop. This includes the test data - if you don't need that, delete the test subdirectory. Make a fresh directory where you want to put stuff from dev.w3.org.
$ cvs -d:pserver:anonymous@dev.w3.org:/sources/public login password? anonymous $ cvs -d:pserver:anonymous@dev.w3.org:/sources/public get 2000/10/swap

From the web

Get the files one by one. cwm.py is the main application file. You can browse the source files on the web, but this is **not** a practical way to install the system.

In the following, we assume $SWAP expands to the place where you have code checked out.

Test Driven Development (Don't trust the docs ;-)

The best test of works is what has been tested. So the list of files in the regression test defines the set of features which are generally checked on each checkin. Cwm developers agree that all the tests have to pass before code is checked in. To run the tests, do make in the swap/test directory. We reckon to add a test for a new bug, so that bugs don't recur in future versions.

Each subdirectory of test has its own detailed.tests file. In that you can put tests for new features. Note that the test commands are all written to be run in $SWAP/test.

How to make a release

1. Remember to cvs update to ensure you have any changes other people have done before running tests.
2. A quick test that your code still works is cd $SWAP/test; make fast
3. The test a release must pass before you make it is cd $SWAP/test; make pre-release
4. Update the releases page with details of the new bug fixes and/or features.
5. Edit $SWAP/Makefile
    - Make sure the HTML files generated from any new .py files are added to the list HTMLS
    - Change the version number if you are going to make a new tarball

Code Overview

Cwm developers agree to keep line lengths below 80 characters, though we have some code that predates that agreement.

llyn.py - The Store

An in-memory store which does the inference. See the Formula class methods for a more or less conventional RDF API. A Forumula is a set of statements.

notation3.py - Serializing/deserializaing RDF/N3

Originally written by Dan Connolly, uses a basic RDF stream parser interface, migrating to API

- Parses N3
- Generates N3

The command line form (alias n3 python notation3.py; n3 -help) allows RDF to be parsed and re-output.

The module will also run as a CGI script to convert N3 to RDF M&S 1.0 - by DanC magic.

- Source

xml2rdf.py Parsing RDF/XML

Based on Python's xmllib, this parser is compatible with the RDF stream interface of, notation3.py. It completes the square of parsers and generators. **Defunct. Now use sax parser and sax2rdf.py.**

- Parses RDF

It has a command line mode for self-test purposes.

- Source

cwm_xxx.py - builtin modules

These are quite easy to add to. Look at a few and clone a similar one to the one you have made.

Design issues

The code above investigated and raised issues discussed in the following documents.

- Notation 3 - an alternative RDF syntax
- Quantification implicit in anonymous nodes

not to mention

- RDFM&S and schema issues
- The question of quoting and BagIDs etc

Acknowledgements

Thanks to Dan Connolly for writing the first code and thereby introducing me to Python, and to him and Sean Palmer and Mark Nottingham for writing built-in function modules. Eric Prud'hommeaux wrote the remote database query and mySQL interface. Sandro Hawke has made various contributions. Yosi Scharf engineered the cwm 1.0.0 release and fixed various bugs and added SPARQL support. Thanks to Sean for his guide to cwm. Thanks for all on #RDFIG for being everything which is #RDFIG.

Development of cwm is supported in part by funding from US Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-00-2-0593, "Semantic Web Development".

License

Cwm: http://www.w3.org/2000/10/swap/doc/cwm.html

Copyright © 2000-2004 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. This work is distributed under the W3C Software License [1] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231

Author: Tim Berners-Lee

10.4 Appendix D

Cwm command line syntax

http://www.w3.org/2000/10/swap/doc/CwmHelp

Cwm Command Line arguments

You can get a list of these by typing cwm --help

Command line RDF/N3 tool
<command> <options> <steps> [--with <more args>]
options:
--pipe Don't store, just pipe out *

steps, in order left to right:

--rdf	Input & Output ** in RDF/XML instead of n3 from now on
--n3	Input & Output in N3 from now on. (Default)
--rdf=flags	Input & Output ** in RDF and set given RDF flags
--n3=flags	Input & Output in N3 and set N3 flags
--ntriples	Input & Output in NTriples (equiv --n3=uspartane -bySubject -quiet)
--language=x	Input & Output in "x" (rdf, n3, etc) --rdf same as: --language=rdf
--languageOptions=y	--n3=sp same as: --language=n3 --languageOptions=sp
--ugly	Store input and regurgitate, data only, fastest *
--bySubject	Store input and regurgitate in subject order *
--no	No output * (default is to store and pretty print with anonymous nodes) *
--base=<uri>	Set the base URI. Input or output is done as though theirs were the document URI.
--closure=flags	Control automatic lookup of identifiers (see below)
<uri>	Load document. URI may be relative to current directory.
--apply=foo	Read rules from foo, apply to store, adding conclusions to store
--patch=foo	Read patches from foo, applying insertions and deletions to store
--filter=foo	Read rules from foo, apply to store, REPLACING store with conclusions
--query=foo	Read a N3QL query from foo, apply it to the store, and replace the store with its conclusions
--sparql=foo	Read a SPARQL query from foo, apply it to the store, and replace the store with its conclusions
--rules	Apply rules in store to store, adding conclusions to store
--think	as -rules but continue until no more rule matches (or forever!)
--engine=otter	use otter (in your $PATH) instead of llyn for linking, etc
--why	Replace the store with an explanation of its contents
--mode=flags	Set modus operandi for inference (see below)
--reify	Replace the statements in the store with statements describing them.
--dereify	Undo the effects of --reify
--flatten	Reify only nested subexpressions (not top level) so that no {} remain.
--unflatten	Undo the effects of --flatten

```
--think=foo             as -apply=foo but continue until no more rule matches (or forever!)
--purge                 Remove from store any triple involving anything in class log:Chaff
--data                  Remove all except plain RDF triples (formulae, forAll, etc)
--strings               Dump :s to stdout ordered by :k wherever { :k log:outputString :s }
--crypto                Enable processing of crypto builtin functions. Requires python crypto.
--help                  print this message
--revision              print CVS revision numbers of major modules
--chatty=50             Verbose debugging output of questionable use, range 0-99
--sparqlServer          instead of outputting, start a SPARQL server on port 8000 of the store

finally:
--with                  Pass any further arguments to the N3 store as os:argv values
        * mutually exclusive
       ** doesn't work for complex cases :-/
Examples:
  cwm --rdf foo.rdf --n3 --pipe     Convert from rdf/xml to rdf/n3
  cwm foo.n3 bar.n3 --think         Combine data and find all deductions
  cwm foo.n3 --flat --n3=spart

Mode flags affect inference extending to the web:
 r   Needed to enable any remote stuff.
 a   When reading schema, also load rules pointed to by schema (requires r, s)
 E   Errors loading schemas of definitive documents are ignored
 m   Schemas and definitive documents loaded are merged into the meta knowledge
     (otherwise they are consulted independently)
 s   Read the schema for any predicate in a query.
 u   Generate unique ids using a run-specific

Closure flags are set to cause the working formula to be automatically expanded to
the closure under the operation of looking up:
 s   the subject of a statement added
 p   the predicate of a statement added
 o   the object of a statement added
 t   the object of an rdf:type statement added
 i   any owl:imports documents
 r   any doc:rules documents
 E   errors are ignored --- This is independent of --mode=E
 e   Smush together any nodes which are = (owl:sameAs)
See http://www.w3.org/2000/10/swap/doc/cwm for more documentation.
Setting the environment variable CWM_RDFLIB to 1 makes Cwm use rdflib to parse
rdf/xml files. Note that this requires rdflib.

Flags for N3 output are as follows:
 a   Anonymous nodes should be output using the _: convention (p flag or not).
d   Don't use default namespace (empty prefix)
e   escape literals --- use \u notation
i   Use identifiers from store - don't regen on output
l   List syntax suppression. Don't use (..)
n   No numeric syntax - use strings typed with ^^ syntax
p   Prefix suppression - don't use them, always URIs in <> instead of qnames.
q   Quiet - don't make comments about the environment in which processing was done.
```

r Relative URI suppression. Always use absolute URIs.
s Subject must be explicit for every statement. Don't use ";" shorthand.
t "this" and "()" special syntax should be suppresed.
u Use \u for unicode escaping in URIs instead of utf-8 %XX
v Use "this log:forAll" instead of @forAll, and "this log:forAll" for "@forSome".
/ If namespace has no # in it, assume it ends at the last slash if outputting.

Flags to control RDF/XML output (after --rdf=) areas follows:
    b - Don't use nodeIDs for Bnodes
    c - Don't use elements as class names
    d - Default namespace suppressed.
    l - Don't use RDF collection syntax for lists
    r - Relative URI suppression. Always use absolute URIs.
    z - Allow relative URIs for namespaces

  Flags to control RDF/XML INPUT (after --rdf=) follow:
    S - Strict spec. Unknown parse type treated as Literal instead of error.
    T - take foreign XML as transparent and parse any RDF in it
        (default it is to ignore unless rdf:RDF at top level)
    L - If non-rdf attributes have no namespace prefix, assume in local <#> namespace
    D - Assume default namespace declared as local document is assume xmlns=""

Note: The parser (sax2rdf) does not support reification, bagIds, or parseType=Literal.
      It does support the rest of RDF inc. datatypes, xml:lang, and nodeIds,

10.5 Appendix E

Cwm builtins

http://www.w3.org/2000/10/swap/doc/CwmBuiltins

Built-in functions

Crypto

@prefix crypto: <http://www.w3.org/2000/10/swap/crypto#>.

CanEncrypt	PublicKeyObjects which are capable of encrypting things
CanSign	PublicKeyObjects which are capable of signing things. True if the algorithm is capable of signing data; false otherwise. To test if a given key object can sign data, use CanSign and HasPrivate.
HasPrivate	Some keys have private parts, some do not. This is the class of those which do.
HashFunction	The cryptographic hash functions are (being functions) unique and are, when secure, assumed unambiguous (the whole point of being hash functions). That is, when you have the right hash, you have the right document. Currently (2001/9) only SHA is given that property.
PublicKeyObject	An object corresponding to a key for some algorithm. The object can hold a public and optionally a private key.
md5	The object is a MD5 hash of the subject.
publicKey	The object is a public key object that doesn't contain the private key data in the subject. This function extracts the public part.
sha	The object is a SHA-1 hash of the subject.
sign	The subject should be a list of two things, a hash string and a key (containing private and public parts). The object is calculated as a signature string by signing the hash with the key's private part.
verify	If the subject is a key object containing private and public parts and the obejct is a list of a hash and a signature, then this is true if and only if the signature is a valid signature of the hash with the key.
verifyBoolean	If the subject is a list containing a keypair, a hash, and a signature, then the object is either "1" if the signature validates or "0" if it does not.

list

@prefix list: <http://www.w3.org/2000/10/swap/list#>.

in	If the object is a list and the subject is in that list, then this is true.
last	If the object is a list and the subject is the last thing that list, then this is true. The last element can be calculated as a function of the list.

log

@prefix log: <http://www.w3.org/2000/10/swap/log#>.

Chaff	Any statement mentioning anything in this class is considered boring and purged by the cwm --purge option. This is a convenience and does not have any value when published as a general fact on the Web.
N3Document	A document which, which parsed as Notation3 as defined in general by http://www.w3.org/DesignIssues/Notation3.html and this schema, conveys the intent of the author by the semantics defined on those specifications, and the semantics defined by the specifications of any other identifiers used in the document.
Truth	Something which is true: believe it as you would believe this. Understood natively by cwm in that it will execute rules in a formula declared a Truth within a formula it is already taking rules from.
conclusion	All possible conclusions which can be drawn from a formula. The object of this function, a formula, is the set of conclusions which can be drawn from the subject formula, by successively applying any rules it contains to the data it contains. This is equivalent to cwm's "--think" command line function. It does use built-ins, so it may, for example, indirectly invoke other documents, validate signatures, etc.
conjunction	"A function to merge formulae: logical AND. The subject is a list of formulae. The object, which can be generated, is a formula containing a copy of each of the formulae in the list on the left. A cwm built-in function.
content	This connects a document and a string that represents it. (Cwm knows how to go get a document in order to evaluate this.) Note that the content-type of the information is not given and so must be known or guessed.
definitiveDocument	When document D is the definitiveDocument for property P, any statement X P Y is true if and only if the semantics of document D include that statement. For example, there may be a definitive document for the zipcode of airports by airport code, and so on. This is useful to let a reasoner know that it can extend its query to the given document. (Cwm will do this if its mode includes "r").
definitiveService	When service S is the definitiveService for property P, any statement X P Y is true iff and only if a query to S returns that it is. The protocol for the service S depends on the scheme. For mysql protocol, the URI of the service is sql://user:password@host.domain/database/. For example, there may be a definitive service for the zipcode of airports by airport code, and so on. This is useful to let a reasoner know that it can help resolve a query by delegating it to the service in question. (Cwm will do this if its mode includes "r").
dtlit	Takes a list of a string and a URI and creates a datatyped literal. For example, { ("2005-03-30T11:00:00" :tz) log:dtlit ?X } => { ?X a :Answer } . will produce "2005-03-30T11:00:00"^^:tz a :Answer .
equalTo	True if the subject and object are the same RDF node (symbol or literal). Do not confuse with daml:EquivalentTo. A cwm built-in logical operator, RDF graph level.
implies	Logical implication. This is the relation between the antecedent (subject) and conclusion (object) of a rule. The application of a rule to a knowledgebase is as follows. For every substitution which, applied to the antecedent, gives a formula which is a subset of the knowledge base, then the result of applying that same substitution to the conclusion may be added to the knowledge base. related: See log:conclusion. (See the CWM manual for command line options to determine how rules from different sources are applied to and the results added to various formula.)
includes	The subject formula includes the object formula. Formula A includes formula B

	if there exists some substitution which when applied to B creates a formula B' such that for every statement in B' is also in A, every variable universally (or existentially) quantified in B' is quantified in the same way in A. Variable substitution is applied recursively to nested compound terms such as formulae, lists and sets. (Understood natively by cwm when in in the antecedent of a rule. You can use this to peer inside nested formulae.)
n3String	The subject formula, expressed as N3, gives this string.
notEqualTo	Equality in this sense is actually the same URI. Do not confuse with daml:EquivalentTo. A cwm built-in logical operator.
notIncludes	The object formula is NOT a subset of subject. True iff log:includes is false. The converse of log:includes. (Understood natively by cwm. The subject formula may contain variables. (In cwm, variables must of course end up getting bound before the log:include test can be done, or an infinite result set would result) Related: See includes
outputString	The subject is a key and the object is a string, where the strings are to be output in the order of the keys. See cwm --strings in cwm --help.
parsedAsN3	The subject string, parsed as N3, gives this formula.
racine	For anything identified by a URI with a frag id, this is the thing identified by the same URI without a hash or frag id. For anything else, it is itself.
rawType	This is a low-level language type, one of log:Formula, log:Literal, log:List, log:Set or log:Other. Example: log:semanticsOrError returns either a formula or a string, and you can check which using log:rawType.
rawUri	This allows one to look at the actual string of the URI which identifies this, for anything, even a blank node or a formula. This peeks into the internal workings of cwm, and so is not normally used. Use log:uri instead.
semantics	The log:semantics of a document is the formula. achieved by parsing representation of the document. For a document in Notation3, log:semantics is the log:parsedAsN3 of the log:contents of the document. For a document in RDF/XML, it is parsed according to the RDF/XML specification to yield an RDF formula (a subclass of N3 log:Formula). [Aside: Philosophers will be distracted here into worrying about the meaning of meaning. At least we didn't call this function "meaning"! In as much as N3 is used as an interlingua for interoperability for different systems, this for an N3 based system is the meaning expressed by a document.] (Cwm knows how to go get a document and parse N3 and RDF/XML it in order to evaluate this. Other languages for web documents may be defined whose N3 semantics are therefore also calculable, and so they could be added in due course. See for example GRDDL, RDF/A, etc)
semanticsOrError	This connects a document and either the formula it parses to, or an error message explaining what went wrong with trying. See log:semantics. (Cwm knows how to go get a document and parse it in order to evaluate this.)
uri	This allows one to look at the actual string of the URI which identifies this. (Cwm can get the URI of a resource or get the resource from the URI.) This is a level breaker, breaking the rule of not looking inside a URI. Use (eg with string:match) to replace RDF's old "aboutEach" functionality. Use to implement the URI spec and protocol specs, etc.

math

@prefix math: <http://www.w3.org/2000/10/swap/math#>.

Function	A math:Function is unique in terms of math:EqualTo.
List	The class of things that are DAML lists were all of the members are math:Value items.
LogicalOperator	a logical operator allows evaluation either way, or testing relationship between two values
ReverseFunction	A math:ReverseFunction is unambiguous in terms of math:EqualTo.
TwoMemberedList	This is the class of things that are math lists with only two members.
Value	The class of things that are numeric float values as in Python.
absoluteValue	The object is calculated as the absolute value of the subject.
atan2	The subject is a pair of numbers. The object is calculated as the arc tangent value of the ratio of the two subject values.
cos	The subject is an angle expressed in radians. The object is calculated as the cosine value of the subject.
degrees	The subject is an angle expressed in radians. The object is calculated as the conversion in degrees of the value of the subject.
difference	The subject is a pair of numbers. The object is calculated by subtracting the second number of the pair from the first.
equalTo	True iff the subject is a string representation of a number which is EQUAL TO a number of which the object is a string representation.
exponentiation	The subject is a pair of numbers. The object is calculated by raising the first number of the power of the second.
greaterThan	True iff the subject is a string representation of a number which is greater than the number of which the object is a string representation.
integerQuotient	The subject is a pair of integer numbers. The object is calculated by dividing the first number of the pair by the second, ignoring remainder.
lessThan	True iff the subject is a string representation of a number which is LESS than a number of which the object is a string representation.
memberCount	The number of items in a list. The subject is a list, the object is calculated as the number of members.
negation	The subject or object is calculated to be the negation of the other.
notEqualTo	True iff the subject is a string representation of a number which is NOT EQUAL to a number of which the object is a string representation.
notGreaterThan	True iff the subject is a string representation of a number which is NOT greater than the number of which the object is a string representation.
notlessThan	True iff the subject is a string representation of a number which is NOT LESS than a number of which the object is a string representation.
product	The subject is a list of numbers. The object is calculated as the arithmetic product of those numbers.
quotient	The subject is a pair of numbers. The object is calculated by dividing the first number of the pair by the second.
remainder	The subject is a pair of integers. The object is calculated by dividing the first number of the pair by the second and taking the remainder.
rounded	The object is calculated as the subject rounded to the nearest integer.
sin	The subject is an angle expressed in radians. The object is calculated as the sine value of the subject.
sinh	The subject is an angle expressed in radians. The object is calculated as the hyperbolic sine value of the subject.
sum	The subject is a list of numbers. The object is calculated as the arithmetic sum of those numbers.
tan	The subject is an angle expressed in radians. The object is calculated as the

	tangent value of the subject.
tanh	The subject is an angle expressed in radians. The object is calculated as the hyperbolic tangent value of the subject.

os

@prefix os: <http://www.w3.org/2000/10/swap/os#>.

argv	The object is looked up as the literal string which was given as the nth command line argument. The os:argv property allows one to make statements whose interpretation is relative to the conditions pertaining at the time of execution. Beware of writing axioms about these, in making optimizations, for example, about reuse of information between runs. The uniqueness of this property pertains to one run of a program. The ont:UniqueProperty may be removed if it messes up more complex processing. (For example, cwm uses a "-with" argument to indicate that the following args should be passed to the RDF system. Example: cwm foo.n3 -think -with bar baz when processing, {"1" os:argv "bar". "2" os:arv "baz"} will be true)
baseAbsolute	The baseAbsolute property of a URI reference string is a string which is an (absolute) URI, generated assuming the base of the running process. This will, for example, generate a file: URI from a unix relative file path when running in file: space. (Cwm uses the current working directory as a base unless the --base option is given)
baseRelative	The baseRelative property of a URI string is a string which is a valid relative form of the URI, relative to the base of the running process. The base of a running unix process is typically a file: URI for the file being processed, or just the current working directory followed by a "/". The relative form is suitable for quotation in a file whose URI is the same (except for anything after the last slash).
environ	The os:environ property of a string is the value corresponding to the string when looked up in the current environment. This is not, of course, something of global significance: it is only used in local processing for passing parameters into a semantic web processor. The subject is the name of the environment variable and the object its value. os:environ is a built-in function in cwm, and corresponds to Python's os.environ[] .

string

@prefix string: <http://www.w3.org/2000/10/swap/string#>.

concat	(obsolete - (was backwards!) - use: string:concatenation)
concatenation	The subject is a list of strings. The object is calculated as a concatenation of those strings.
contains	True if the subject string contains the object string.
containsIgnoringCase	True if the subject string contains the object string, with the comparison done ignoring the difference between upper case and lower case characters.
endsWith	True if the subject string ends with the object string.
equalIgnoringCase	True if the subject string is the same as object string ignoring differences between upper and lower case.
greaterThan	True if the string is greater than the object when ordered according to Unicode(tm) code order.
lessThan	True if the string is less than the object when ordered according to Unicode(tm) code order.
matches	The subject is a string; the object is a regular expression in the perl, python style. It is true iff the string matches the regexp.

notEqualIgnoringCase	True if the subject string is the NOT same as object string ignoring differences between upper and lower case.
notGreaterThan	True if the string is NOT greater than the object when ordered according to Unicode(tm) code order.
notLessThan	True if the string is NOT less than the object when ordered according to Unicode(tm) code order.
notMatches	The subject string; the object is is a regular expression in the perl, python style. It is true if the string does NOT match the regexp.
scrape	The subject is a list of two strings. The second string is a regular expression in the perl, python style. It must contain one group (a part in parentheses). If the first string in the list matches the regular expression, then the object is calculated as being the part of the first string which matches the group.
startsWith	True if the subject string starts with the object string.

time

@prefix time: <http://www.w3.org/2000/10/swap/time#>.

day	For a date-time, its time:inSeconds is the (string representation of) the two-digit day of the month. Cwm implements this as a function.
dayOfWeek	For a date-time, its time:dayOfWeek is the (string representation of) the day number within the week, Sunday being 0. Currently the result is a single digit string but do not count on it being anything other than a valid integer representation. Cwm implements this as a function.
gmTime	For a date-time format string, its time:gmtime is the result of formatting the Universal Time of processing in the format given. If the format string has zero length, then the ISOdate standard format is used. [is time:gmtime of ""] is therefore the current date time. It will end with "Z" as a timezone code. Cwm implements this as a function. Rules which use this function will of course NOT be repeatable.
hour	For a date-time, its time:inSeconds is the two-digit hour in the 24 hour clock. Cwm implements this as a function.
inSeconds	For a date-time, its time:inSeconds is the (string representation of) the floating point number of seconds since the beginning of the era on the given system. Do not assume a particular value, always test for it. Cwm implements this as a bidirectional function: you can calculate the ISO date from the seconds since the beginning of the era, or vice-versa.
localTime	For a date-time format string, its time:localTime is the result of formatting the current time of processing and local timezone in the format given. If the format string has zero length, then the ISOdate standard format is used. [is time:localTime of ""] is therefore the current date time. It will end with a numeric timezone code or "Z" for UTC (GMT). Cwm implements this as a function. Rules which use this function will of course NOT be repeatable.
minute	For a date-time, its time:minute is the two-digit number of seconds. Cwm implements this as a function.
month	For a date-time, its time:inSeconds is the two-digit month. Cwm implements this as a function.
second	For a date-time, its time:second is the two-digit number of seconds. Cwm implements this as a function.
timeZone	For a date-time, its time:timeZone is the trailing timezone offset part, e.g. "-05:00". Cwm implements this as a function.
year	For a date-time, its time:inSeconds is the (string representation of) the four-digit year. Cwm implements this as a function.

11.0 List of Symbols, Abbreviations and Acronyms

awk: a general purpose computer language that is designed for processing text-based data, either in files or data streams.

Bnode: stands for "blank node", which refers to the fact that the corresponding nodes in the RDF graph are "blank", i.e., have no label.

CSAIL: (Computer Science and Artificial Intelligence Laboratory), an interdisciplinary research laboratory at MIT formed by the merger of the Laboratory for Computer Science (LCS) and Artificial Intelligence Laboratory.

CWA: (Closed World Assumption) the presumption that what is not currently known to be true is false.

cwm: a general-purpose data processor for the Semantic Web.

DAML: (DARPA agent markup language) aims to enable the next generation of the web that moves from simply displaying content to one that actually understands the meaning of the content.

DARPA: (Defense Advanced Research Projects Agency), an agency of the United States Department of Defense responsible for the development of new technology for use by the military.

DIG: (Decentralized Information Group), a research project based at MIT/CSAIL to explore technical, institutional and public policy questions necessary to advance the development of global, decentralized information environment

DL: (Description Logic) a family of knowledge representation languages which can be used to represent the terminological knowledge of an application domain in a structured and formally well-understood way

GPS: (Global Positioning System), satellites broadcast precise timing signals by radio to GPS receivers, allowing them to accurately determine their location with longitude, latitude and altitude.

grep: a command line utility originally written for use with the Unix operating system.

HTML: (Hypertext Markup Language) a markup language designed for the creation of web pages with hypertext and other information to be displayed in a web browser.

HTTP: (Hypertext Transfer Protocol) an open internet protocol used to provide a way to publish and receive HTML pages.

IETF: (Internet Engineering Task Force), a standards organization that develops and promotes Internet standards; in particular those of the TCP/IP protocol suite.

IMAP: (Internet Message Access Protocol), an application layer Internet protocol that allows a local client to access e-mail on a remote server.

IRC: (Internet Relay Chat), a form of instant communication over the Internet.

LCS: (Laboratory for Computer Science), a research laboratory at MIT (now part of CSAIL).

LISP: a family of programming languages, the name Lisp derives from "List Processing". Linked lists are one of Lisp languages' major data structures, and Lisp source code is itself made up of lists.

MIT: Massachusetts Institute of Technology

MySQL: a multithreaded, multi-user, SQL Database Management System

N3/Notation3: a language which is a compact and readable alternative to RDF's XML syntax, but also is extended to allow greater expressiveness.

OIL: (Ontology Inference Layer or Ontology Interchange Language), an Ontology Infrastructure for the Semantic Web.

OWL: (Web Ontology Language), a markup language for publishing and sharing data using ontologies on the Internet. It is a vocabulary extension of RDF and is derived from the DAML+OIL Web Ontology Language

OFX: (Open Financial Exchange), a format starting to be deployed by banking institutions to provide customer's data upon request**.**

OS: (Operating System), a software program that manages the hardware and software resources of a computer.

PAW: (Policy Aware Web) a rule-based policy management system to provide a scalable mechanism for the exchange of rules and, eventually proofs, for access control on the Web.

PDA: (Personal Digital Assistant), a digital device which can include the functionality of a computer, a cellphone, a music player and a camera.

PINS: (Personal INformation Schema), tools to enable users to attach usage restrictions to use and re-use of information, especially personalized information that they contribute to the Semantic Web

QIF: (Quicken Interchange Format), a specially formatted American Standard Code for Information Interchange (ASCII) text file. It is used to transfer data between different Quicken data files, from a financial institution's Web site to Quicken, and in some cases, from other financial programs.

RDF: (Resource Description Framework), a family of specifications originally designed as a metadata model using XML but which has come to be used as a general method of modeling knowledge.

RIF: (Rule Interchange Format), a Semantic Web language to enable interchange of rules.

RuleML: (Rule Markup Language), a markup language which permits both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks.

RSS: (Really Simple Syndication), a form of web syndication used by news websites and weblogs

sed: (Stream EDitor) is a simple but powerful computer program used to apply various pre-specified textual transformations to a sequential stream of text data.

SPARQL: (Simple Protocol and RDF Query Language), a query language and data access protocol for the Semantic Web.

SQL: (Structured Query Language), a computer language used to create, modify, retrieve and manipulate data from relational database management systems.

SW: (Semantic Web), a project to create a universal medium for information exchange by putting documents with computer-processable meaning (semantics) on the World Wide Web.

SWeLL: (Semantic Web Logic Language), a proposed logic language for use on an open and unbounded Semantic Web which was delivered as N3 (Notation3).

TAMI: (Transparent Accountable Datamining Initiative), technical, legal, and policy foundations for transparency and accountability in large-scale aggregation and inferencing across heterogeneous information systems.

Turtle: (Terse RDF Triple Language), a text syntax for RDF.

URI: (Uniform Resource Identifier), a short string of characters that comprises a name or address that can be used to refer to a resource. It is a fundamental component of the World Wide Web.

W3C: (World Wide Web Consortium) an international consortium where member organizations, a full-time staff, and the public, work together to develop standards for the World Wide Web.

WWW: (World Wide Web), a global, read-write information space.

XML: (Extensible Markup Language (XML), general-purpose markup language for creating special-purpose markup languages, capable of describing many different kinds of data.

XSLT: (Extensible Stylesheet Language Transformations) is an XML-based language used for the transformation of XML documents.